

REAL-TIME REGISTRATION AND SIMULATION IN MEDICAL IMAGING

RAMTIN SHAMS

RESEARCH SCHOOL OF INFORMATION SCIENCES AND ENGINEERING
COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

OCTOBER 2009

A THESIS SUBMITTED FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
THE AUSTRALIAN NATIONAL UNIVERSITY



THE AUSTRALIAN NATIONAL UNIVERSITY

© RAMTIN SHAMS, 2009.

PRODUCED IN L^AT_EX 2_ε.


To my beautiful wife for the meaning she brings into my life and to my loving parents for the passion they bestowed on me to embark on the path to learning.

Declaration

The contents of this thesis are the results of original research and have not been submitted for a higher degree to any other university or institution.

The majority of the work in this thesis has been published as journal papers or conference proceedings.

The research work presented in this thesis was carried out in collaboration with my supervisors and other researchers as noted in relevant publications. However, the substantial majority of this work is my own.



Ramtin Shams

October 2009

Research School of Information Sciences and Engineering,
The Australian National University,
Canberra,
ACT 0200,
Australia

Acknowledgements

This PhD has been quite a journey, the pursuit of which took me to three continents. It gave me the opportunity to collaborate with some of the most exceptionally talented individuals, the list of whom is delightfully long. I intend to thank them each and individually in the following lines; for they have been part of an experience that has been as rewarding as it has been unique.

First and foremost, I am grateful beyond description to my supervisors, Prof. Kennedy and Prof. Hartley. I have been privileged to have had the opportunity to work closely with not one but two incredibly talented and accomplished scientists. I am thankful to Prof. Kennedy for pointing me in the right direction from the very beginning and for being my mentor.

I am most thankful for the support I received at ANU. I was given the liberty to freely pursue my research interests. My publications and international visits were made possible by the generous financial support through VC travel grants, departmental funds, and my supervisors' research funds.

I am utterly grateful to Dr. Sadeghi for teaching me the alphabet of research and for many long hours of discussion and scientific exchange. I am also grateful to Dr. Barnes for fruitful discussions and exchange of ideas. They are both extremely bright scientists and it has been a privilege to work with them.

I cannot thank Prof. Navab enough for his support, for his open approach towards research, and making me feel part of his team during my time at TUM. He has created a most wonderful, vibrant, friendly, and successful research environment. I had a most enjoyable time, both personally and professionally, among the friendly folks at CAMP in Munich. Particularly, I wish to thank Oliver Kutter and Christian Wachinger for going out of their way to make my visit a memorable one. They are both extremely intelligent researchers and I greatly enjoyed collaborating with them.

I would like to thank A/Prof Abolmaesoumi and A/Prof Ourselin for giving me an introduction to the medical image analysis discipline and community. I am sincerely grateful to A/Prof Abolmaesoumi for many helpful discussions and

suggestions regarding my research during our meetings in MICCIA 2008 and 2009.

I wish to thank Dr. Vosburgh for his support of my research and allowing me to be part of his team at CIGL at Harvard Medical School. It has been an honor to work with such a versatile, dedicated, and experienced scientist. It was also a great pleasure to collaborate with Dr. San Jose Estepar and Dr. Patil during my time in Boston.

A special thanks to RSISE staff. Our kind department admin Elspeth Davies, our hardworking grant officers Ewa Ziolkowska and Eva Lerable, our ever-helpful IT team, Nic Boling, Peter Shevchenko, Stuart Watson and particularly James Ashton. Some of my publications only made it in time because they gladly burdened themselves with my research ideas from custom-built hardware to temporary clusters and never said no to a challenge.

I cannot possibly thank my loving parents, Fakhri and Firooz, enough for all the sacrifices that they have made over the years so that I can fulfil my dreams. I am most grateful to them for instilling in me the fire of learning from an early age and one that I will carry to the end.

Above all, I am indebted to the love of my life, Parastoo. She was my inspiration to rise to this challenge. She held my hand every step of the way and gave me strength and kept me going through the many ups and downs. Without her love and support, I would not have gotten far.

Abstract

Advances in imaging technologies have resulted in the availability of a large array of medical scanners for clinical applications which produce images of differing *modalities*. As a consequence, the last decade has seen a dramatic increase in the volume of information generated by radiology and pathology departments. This has resulted in an increasing need for automated and semi-automated tools for analysis of medical data, in order to put this information into better use.

Modern medical imaging technologies are capable of producing high resolution 3D or 4D (3D + time) images. This makes medical image processing tasks at least one dimension more compute-intensive than standard 2D image and video processing applications. In this thesis, we look at methods for improving the performance and robustness of *registration* of medical images and real-time synthesis of ultrasound. We use two principal methods to improve the computational efficiency of medical image analysis tasks through (a) algorithmic and methodic enhancements and (b) use of high performance and massively parallel processing architectures.

We first look at improving the robustness and computational efficiency of *rigid* and *similarity* registration of multi-modal images through (a) systematic reduction of dimensionality of the data-sets and (b) decoupling estimation of registration parameters that leads to a reduction of the complexity of the search space. We then present methods suitable for *collinear* and *deformable* registration of images specifically designed for the massively parallel architecture of the modern graphics processing units (GPUs).

In the computing domain, a paradigm shift is happening as a result of the introduction of many-core processors and massively multi-processing platforms. Tera-flop performance is now available on single-chip commodity GPUs. Moreover, GPUs are no longer limited to graphics applications, but are emerging as usable general purpose computing devices. These systems, plus other *accelerator* technologies (such as the STI Cell Broadband Engine, upcoming Intel Larrabee processor, FPGAs and DSPs) are already making many computational problems that were previously reserved for super-computing systems, solvable on desktop computers,

at a minute fraction of the price, and with significantly lower power requirements. The advent of this new generation of low-cost high performance computing platforms presents both numerous opportunities and challenges. We hope to convince the reader, through our investigation of the registration of medical images on the GPU, that adaptation of existing algorithms for massively multiprocessing environments is a non-trivial task that often involves reinventing and rethinking existing methods from grounds up and specifically designing them for parallel computation on thousands of concurrent threads. We also demonstrate that, when properly executed, GPU adaptation of algorithms can result in significant savings in computation times.

In the final chapter of the thesis we investigate real-time synthesis of ultrasound from tomographic modalities which can be used in registration of ultrasound with other modalities and in training simulators. The main idea is to reduce the computational cost by devising a simple acoustic model that can sufficiently represent ultrasonic effects for the task at hand. We further improve the computation time by a GPU-based implementation of the model and demonstrate a simulation and visualization software that achieves interactive frame rates.

Publications

Related Publications

Journal Articles

1. **R. Shams**, P. Sadeghi, R. A. Kennedy, and R. Hartley, "A survey of medical image registration on multi-core and GPU," *IEEE Signal Processing Mag.* (to appear), Mar. 2010.
2. **R. Shams**, P. Sadeghi, R. A. Kennedy, and R. Hartley, "Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images," *Computer Methods and Programs in Biomedicine* (accepted), Nov. 2009.
3. O. Kutter, **R. Shams**, and N. Navab, "Visualization and GPU-accelerated simulation of medical ultrasound from CT images," *Computer Methods and Programs in Biomedicine*, vol. 94, no. 3, pp. 250–266, June 2009.

Conference Papers

4. **R. Shams**, R. S. J. Estepar, V. Patil, and K. G. Vosburgh, "Intraoperative ultrasound probe calibration in a sterile environment," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI) workshop on Augmented Environments for Medical Imaging and Computer-aided Surgery*, London, UK, Sept. 2009, pp. 53–60.
5. **R. Shams**, R. Hartley, and N. Navab, "Real-time simulation of medical ultrasound from CT images," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, New York, USA, Sept. 2008, pp. 734–741.
6. C. Wachinger, **R. Shams**, and N. Navab, "Estimation of acoustic impedance from multiple ultrasound images with application to spatial compounding," in

IEEE Computer Society Workshop on Mathematical Methods in Biomedical Image Analysis (MMBIA), Anchorage, Alaska, June 2008.

7. **R. Shams** and N. Barnes, "Speeding up mutual information computation using NVIDIA CUDA hardware," in *Proc. Digital Image Computing: Techniques and Applications (DICTA)*, Adelaide, Australia, Dec. 2007, pp. 555–560.
8. **R. Shams**, N. Barnes, and R. Hartley, "Image registration in Hough space using gradient of images," in *Proc. Digital Image Computing: Techniques and Applications (DICTA)*, Adelaide, Australia, Dec. 2007, pp. 226–232.
9. **R. Shams** and R. A. Kennedy, "Efficient histogram algorithms for NVIDIA CUDA compatible devices," in *Proc. Int. Conf. on Signal Processing and Communications Systems (ICSPCS)*, Gold Coast, Australia, Dec. 2007, pp. 418–422.
10. **R. Shams**, R. A. Kennedy, P. Sadeghi, and R. Hartley, "Gradient intensity-based registration of multi-modal images of the brain," in *Proc. IEEE Int. Conf. on Computer Vision (ICCV)*, Rio de Janeiro, Brazil, Oct. 2007.
11. **R. Shams**, R. A. Kennedy, and P. Sadeghi, "Efficient image registration by decoupled parameter estimation using gradient-based techniques and mutual information," in *Proc. IEEE Region 10 Conf. (TENCON)*, Taipei, Taiwan, Oct. 2007.
12. **R. Shams**, P. Sadeghi, and R. A. Kennedy, "Gradient intensity: A new mutual information based registration method," in *Proc. IEEE Computer Vision and Pattern Recognition (CVPR) Workshop on Image Registration and Fusion*, Minneapolis, MN, June 2007.

Other Publications

During my PhD I contributed and published in areas other than Medical Imaging as listed below. These contributions have not been directly referenced or included as part of this thesis.

Journal Articles

13. P. Sadeghi, **R. Shams**, and D. Traskov, "An optimal adaptive network coding scheme for minimizing decoding delay in broadcast erasure channels,"

EURASIP J. Wireless Commun. and Networking (to appear), Apr. 2010.

14. P. Sadeghi, P. O. Vontobel, and **R. Shams**, "Optimization of information rate upper and lower bounds for channels with memory," *IEEE Trans. Inform. Theory*, vol. 55, no. 2, pp. 663–688, Feb. 2009.
15. P. Sadeghi, R. A. Kennedy, P. Rapajic, and **R. Shams**, "Finite-state Markov modeling of fading channels: A survey of principles and applications," *IEEE Signal Processing Mag.*, vol. 25, no. 5, pp. 57–80, Sept. 2008.

Conference Papers

16. **R. Shams** and P. Sadeghi, "Bar code recognition in highly distorted and low resolution images," in *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 1, Honolulu, HI, Apr. 2007, pp. I-737–I-740.
17. P. Sadeghi, P. O. Vontobel, and **R. Shams**, "Optimizing information rate bounds for channels with memory," in *Proc. IEEE Int. Symp. on Inform. Theory (ISIT)*, Nice, France, June 2007.

List of Acronyms

ALU	Arithmetic Logic Unit
API	Application Programming Interface
CAL	Compute Abstraction Layer
CC	Correlation Coefficient
CCD	Charge-Coupled Device
CPU	Central Processing Unit
CR	Correlation Ratio
CT	Computed Tomography
CUDA	Compute Unified Device Architecture
DM	Distributed Memory
DSM	Distributed Shared Memory
DVR	Direct Volume Rendering
ECC	Error Correcting Code
EM	Electromagnetic
EUS	Endoscopic Ultrasound
FLOPS	Floating Point Operations per Second
FPGA	Field Programmable Gate Array
GI	Gradient Intensity
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HPC	High Performance Computing
IGT	Image-Guided Therapy
LUS	Laparoscopic Ultrasound
MI	Mutual Information
MIRIT	Multi-modality Image Registration using Information Theory
MMP	Massively Multiprocessing
MPI	Message Passing Interface
MPR	Multi-Planar Reconstruction

NCC	Normalized Cross Correlation
NMI	Normalized Mutual Information
NUMA	Non-Uniform Memory Access
pdf	Probability Density Function
PET	Positron Emission Tomography
PI	Pixel Intensity
pmf	Probability Mass Function
PPE	Power Processor Element
PV	Partial Volume
RAM	Random Access Memory
RMS	Root Mean Square
SDK	Software Development Kit
SIFT	Scale-Invariant Feature Transform
SIMD	Single Instruction Multiple Data
SIMT	Single Instruction Multiple Threads
SMP	Symmetric Multiprocessing
SNR	Signal-to-Noise Ratio
SPE	Synergistic Processor Element
SSD	Sum of Squared Differences
SSE	Streaming SIMD Extensions
TRE	Target Registration Error
US	Ultrasound
UVH	Uniform Volume Histogram

Contents

Dedication	i
Declaration	iii
Acknowledgements	v
Abstract	vii
Publications	ix
List of Acronyms	xiii
1 Introduction	1
1.1 Multiprocessing in an Operating Room	1
1.2 Outline	3
1.3 Summary of the Contributions	4
2 Image Registration	7
2.1 Problem Statement	8
2.2 Transformer	9
2.2.1 Rigid	11
2.2.2 Similarity	11
2.2.3 Affine	11
2.2.4 Projective	12
2.2.5 B-Spline	12
2.3 Similarity/Distance Measures	13
2.3.1 Sum of Squared Differences (SSD)	14
2.3.2 Correlation Coefficient (CC)	15
2.3.3 Mutual Information (MI)	15
2.3.4 Correlation Ratio (CR)	16
2.3.5 Probability Distribution Estimation	17

2.3.6	Comparison of Measures	17
2.4	Optimizer	18
2.4.1	Powell	21
2.4.2	Simplex	21
2.4.3	Soblex	22
2.4.4	Gradient Descent	23
2.4.5	Functional Optimization Using Variational Calculus	23
3	Gradient Intensity-Based Registration	27
3.1	Gradient Intensity - The Concept	30
3.1.1	Gradient Field of an Image	30
3.1.2	Gradient Intensity for 2D Images	32
3.1.3	Gradient Intensity for 3D Images	33
3.1.4	Entropy and MI of Gradient Intensity	35
3.1.5	An Alternative Interpretation of 2D Gradient Intensity Based on Hough Transform	36
3.1.6	Uniform Volume Histogram	39
3.2	2D Registration	43
3.2.1	Estimating the Rotation Parameter	43
3.2.2	Estimation of Scale and Translation	44
3.2.3	Enhanced Powell Optimization	46
3.3	Experiments	47
3.3.1	Registration of 2D Images	47
3.3.2	Registration of Multi-Modal Images	48
3.3.3	Discussion	50
3.4	2D Registration in Hough Space	52
3.4.1	Estimation of Transformation Parameters	52
3.4.2	Final Optimization	57
3.4.3	Experiments	57
3.4.4	Discussion	61
3.5	3D Registration	63
3.5.1	Estimating Rotation Parameters	63
3.5.2	Soblex Optimization	64
3.5.3	Estimating Translation Parameters	64
3.5.4	Final Optimization	65
3.5.5	Results	66
3.5.6	Discussion	70

4	Registration on High Performance Computing Architectures	73
4.1	Multi-CPU Implementations	74
4.1.1	Symmetric Multiprocessing	74
4.1.2	Multiprocessing with Non-Uniform Memory Access	75
4.1.3	Multiprocessing with Distributed Memory	76
4.2	Accelerator Implementations	77
4.2.1	Cell Broadband Engine	77
4.2.2	Field Programmable Gate Array (FPGA)	78
4.2.3	Graphics Processing Unit (GPU)	78
4.3	Massively Multiprocessing on GPUs	80
4.3.1	An Overview of CUDA	81
4.4	Summary of the Literature	87
5	Registration on the GPU	91
5.1	Parallel Histogram Computation	92
5.2	Parallel Histogram Computation on the CPU	93
5.2.1	Parallelization with Atomic Operations	94
5.2.2	Synchronization-Free Parallelization	95
5.3	Histogram Computation on the GPU	96
5.3.1	Method 1: Simulating Atomic Updates in Software	98
5.3.2	Method 2: Synchronization-Free Parallelization	102
5.3.3	Method 3: Approximate Histogram	104
5.3.4	Method 4: Sort and Count	106
5.3.5	Comparison of Histogram Computation Methods	114
5.3.6	Computation of Joint Histograms	116
5.4	Parallel Registration	116
5.4.1	MI Computation	117
5.4.2	Experiments: Approximate Histogram	119
5.4.3	Experiments: Bitonic Sort and Count	121
5.4.4	Fast Deformable Registration	125
5.5	Discussion	127
6	Real-time Simulation and Visualization of Ultrasound	131
6.1	Introduction	131
6.2	Method	133
6.2.1	A Simple Acoustic Model for Ultrasound	133
6.2.2	Creating the Reflection Image	136
6.2.3	Creating the Scattering Image	137

6.2.4	Creating the Ultrasound Image	139
6.3	Visualization and GPU-Accelerated Simulation of Ultrasound . . .	140
6.3.1	GPU-Accelerated Ultrasound Simulation	141
6.3.2	The Simulation Pipeline	143
6.3.3	Real-Time Visualization	145
6.3.4	Visual Consistency of Ultrasound Rendering	148
6.3.5	Visualization Pipeline	148
6.3.6	User Interface	149
6.3.7	Computational Performance	152
6.3.8	Performance of Simulation and Visualization in Training Ap- plications	153
6.3.9	Performance of Simulation for Registration Applications . .	154
6.4	Discussion	158
7	Conclusions	165
Appendices		
Appendix A	Rotation Matrix	167
Appendix B	Medical Imaging Coordinate Systems	169
B.1	Anatomical Directions and Planes	169
B.2	Voxel Ordering	169
B.3	Subject Coordinate System	170
B.4	2D and 3D Display of Anatomical Planes	171
Appendix C	Hardware Configuration	173
Appendix D	Intraoperative Ultrasound Probe Calibration in a Ster- ile Field	175
D.1	Introduction	175
D.2	Air Calibration	177
D.2.1	Mold Calibration	178
D.2.2	Funnel Calibration	179
D.2.3	Closest Point Calibration	179
D.3	Results	183
D.4	Discussion	183
Bibliography		185

Chapter 1

Introduction

... *In an operating room not so far into the future ...*

A surgeon is performing a potentially life-saving *pancreatectomy* on a patient in early stages of pancreatic cancer. Two small incisions of no more than half an inch allow *laparoscopic* tools including a video camera and an ultrasound probe to be guided inside the abdominal cavity. A third, larger incision, is occupied by a hand-access device that facilitates the operation. The surgeon is able to locate the tumor in the ultrasound view with ease. This is largely possible due to a newly installed 3D navigation and visualization system that virtually renders the patient transparent.

The visualization system combines data from preoperative magnetic resonance (MR) and computed tomography (CT) scans with intraoperative laparoscopic ultrasound data to produce real-time high quality and dynamic 3D images of the patient, in a process better known as *multi-modal registration* and *fusion*. The high quality 3D images of the tumor and the surrounding tissue allow the surgeon to resect the malignant cells with little damage to healthy structures.

Such a minimally invasive approach avoids the trauma of open surgery, and a faster recovery time means that the patient will be released from the hospital in just two days.

1.1 Multiprocessing in an Operating Room

Image-guided therapy (IGT) systems play an increasingly important role in clinical treatment and interventions. By providing more accurate information about a patient during a procedure, these systems improve the quality and accuracy of procedures and make less invasive options for treatment available. They contribute to reduced morbidity rate, intervention time, post-intervention care, and procedure

costs. For practical reasons, however, imaging systems that can be deployed in an operating room produce images with lower resolutions and lower signal to noise ratios than can be achieved by the state-of-the-art imaging systems preoperatively. Therefore, it is desirable to be able to use preoperative images of a patient together with those acquired during a procedure for best results. In brain surgery, for example, the main challenge is to remove as much of the malignant tissue as possible without affecting critical structures and while minimizing damage to healthy tissue. The surgeon uses high quality CT and MR scans of the patient to carefully plan a procedure. During a procedure, however, the brain undergoes varying levels of deformations at different stages of the surgery known as the *brain shift*. This brain shift, a result of change in the intracranial pressure, leakage of cerebrospinal fluid and removal of tissue, affects the accuracy of earlier planning and needs to be compensated for. The surgeon may take a number of intraoperative scans to correct the plan based on patient's current state and also to detect complications such as bleeding. To support the surgeon, the IGT system needs to register intraoperative scans with the patient and with preoperative images.

Modern medical imaging technologies are capable of producing high resolution 3D or 4D (3D + time) images. This makes medical image processing tasks at least one dimension more compute-intensive than standard 2D image processing applications. The higher computational cost of medical image analysis together with the time constraints imposed by the medical procedure determine the range of tools that can be practically offered through an IGT platform. It also often means that an IGT platform has to rely on high performance computing (HPC) hardware and highly parallelized software. There are other practical considerations. For example, equipment used in an operating room should be designed to minimize footprint, power consumption, operating noise, and cost.

An IGT system needs to carry out its tasks in a surgically acceptable time to minimize their impact on the clinical procedure. Therefore, computational efficiency is an important requirement for algorithms that are designed for use in a surgical setup. The efficiency of algorithms can be improved through methodical and computational enhancements. Methodical improvements involve designing algorithms with inherently lower computational complexity, whereas computational improvements involve adaptation of methods for parallelization and optimal execution on a specific target platform. In this thesis, we look at both avenues for improving the efficiency of ultrasound simulation and certain image registration tasks.

The continued development of multi-core and massively multiprocessing archi-

tructures in recent years holds great promise for interventional setups. In particular, massively multiprocessing graphics units with general purpose programming capabilities have emerged as front runners for low cost high performance processing. HPC, in the order of 1 TFLOPS, is available on commodity single-chip graphics processing units (GPUs) with power requirements not much greater than an office computer. Multi-GPU systems with up to 8 GPUs can be built in a single host and can provide a nominal processing capacity of 8 TFLOPS with less than 1500W power consumption under full load.

Hardware and architectural complexities in designing multi-core systems aside, perhaps as big a challenge is an overhaul of existing application design methodologies to allow efficient implementation on a range of massively multi-core architectures. As one quickly might find, direct adaptation of existing serial algorithms is more often than not neither possible due to hardware constraints nor computationally justified.

1.2 Outline

We discuss the registration problem in Chapter 2 where we define the basic concepts of image registration and its main components: (a) a transformation model, (b) a measure of distance or similarity, and (c) an optimization strategy.

In Chapter 3, we look at improving the robustness and computational efficiency of *rigid* and *similarity* registration of multi-modal images through (a) systematic reduction of dimensionality of the data-sets and (b) decoupling estimation of registration parameters that leads to a reduction of the complexity of the search space.

We discuss parallelization of registration methods in Chapter 4, where we give an overview of high performance computing platforms with an emphasis on aspects that are most relevant to image registration tasks. Then in Chapter 5, we apply these parallelization concepts to develop methods suitable for *collinear* and *deformable* registration of images specifically designed for the massively parallel architecture of the modern graphics processing units (GPUs).

In the final chapter of the thesis we investigate real-time synthesis of ultrasound from tomographic modalities which can be used in registration of ultrasound with other modalities and in training simulators. The main idea is to reduce the computational cost by devising a simple acoustic model that can sufficiently represent ultrasonic effects for the task at hand. We further improve the computation time by a GPU-based implementation of the model and demonstrate a simulation and visualization software that achieves interactive frame rates.

1.3 Summary of the Contributions

A list of the main contributions of this thesis follows:

- **Gradient Intensity**, Section 3.1.2, Section 3.1.3

We introduce the concept of *gradient intensity* (GI), a measure of directional strength of an image based on its gradient content. We show that the mutual information (MI) of GI can be used as a measure for registration and that it reduces the dimensionality of similarity registration and allows for decoupled estimation of the registration parameters.

- **Soblex Optimization**, Section 2.4.3, Section 3.5.2

We introduce a quasi-global optimization method based on simplex and sampling of the parameter space with a Sobol quasi-random sequence. The method is called Soblex and allows for an easy trade-off between the robustness and efficiency of the optimization. It is particularly suited to finding a global minimum in the presence of several local minima.

- **Uniform Volume Histogram**, Section 3.1.6

We propose a computationally efficient method for histogram computation which estimates the probability density of the underlying data with little dependency on the number of bins. We also demonstrate that the method is resistant to noise and results in smoother cost functions when used in conjunction with a GI-based measure.

- **Enhanced Powell Optimization**, Section 3.2.3

We improve the convergence rate of Powell's optimization algorithm by introducing a set of resolution parameters. The resolution parameters are expressed in units of each parameter being optimized and can be more naturally defined than standard parameters used in Powell's convergence such as the fractional tolerance and absolute tolerance.

- **Hough Transform Formulation of the Registration Problem**, Section 3.1.5, Section 3.4

We give an alternative interpretation of the GI based on a Hough transform is given. We formulate the registration problem in the Hough domain and show that the transformation can be used for robust and efficient registration of images.

- **Robust Estimation of Translation Parameters Using an Inverse Radon Back-Projection**, Section 3.4.1

We propose a robust method for estimation of translation parameters in Hough space which is similar to an inverse Radon back projection. The method allows for estimation of parameters from as low as 8 projected lines in a 2D space.

- **Classification and Comparison of Image Registration Methods for HPC Architectures**, Chapter 4

We look at previous, recent, and state-of-the-art methods for registration of medical images on a range of HPC architectures including symmetric multiprocessing (SMP), massively multiprocessing (MMP) and architectures with distributed memory (DM) and non-uniform memory access (NUMA). We define and describe concepts of interest in the context of image registration and high performance computing. We highlight registration related issues as we explore the HPC problem domain. This approach presents a fresh angle on the subject than previously investigated by the more general and classic reviews of registration methods. We have also endeavored to provide a comprehensive summary of existing contributions from various groups and to normalize their results for comparison of different methods and technologies.

- **Histogram Computation on the GPU**, Section 5.3.1, Section 5.3.2, Section 5.3.3, Section 5.3.4

We present several methods for efficient computation of histograms suitable for the streaming architecture of massively multiprocessing GPUs. The histogram computation methods play an important role in efficient computation of MI-based registrations on the GPU.

- **Sort and Count Algorithm**, Section 5.3.4

We present a novel approach for parallelization of histograms that does not require synchronization or use of atomic operations. The method is based on sorting a sequence while computing the number of similarly-valued elements at the same time. We specify the appropriate class of sort algorithms that can be used for this purpose and provide a GPU-based implementation of the method.

- **Real-Time Registration of Images in the Vanderbilt Database**, Section 5.4.3

We demonstrate that using the sort and count method, the multi-modal image pairs from the Vanderbilt database can be registered in less than one second on a commodity GPU.

- **Fast Deformable Registration on the GPU**, Section 5.4.4

We present a fast deformable B-spline type registration for multi- and mono-modality images using the histogram computation methods described in Chapter 5 on a commodity GPU.

- **Ray-Based Model of Ultrasound**, Section 6.2

We develop an acoustic model that can be used in real-time for simulation of large-scale reflections, attenuation due to reflections, effect of a finite beam-width, and view-dependent shadow and occlusion effects in an ultrasound image.

- **Real-Time Simulation of Ultrasound on the GPU¹**, Section 6.3

We present a fast GPU-based method for simulation of ultrasound images from volumetric CT scans and their visualization. We show that the ultrasound simulation problem can be formulated as a ray casting problem. We use the graphics API for our GPU-based ultrasound simulation. Use of the graphics API ensures that our implementation is relatively independent of the graphics hardware and can be run on a wider range of devices.

¹This is a joint work with Oliver Kutter (Technische Universität München) and was carried out during my visit to Munich.

Chapter 2

Image Registration

Image *registration* is a problem commonly encountered in computer vision and particularly in medical image analysis. Registration is a process that aims to find the optimal transformation that best aligns two or more corresponding images. The main motivation for registration is the expectation that one can draw more useful conclusions by combining and comparing information from various sources, subjects, viewing angles, and at different times. To draw meaningful conclusions, one needs to express the information in a common framework. In the context of images, this common framework is provided by image registration. Registration is often a precursor to data *fusion*, *segmentation*, and *labeling*.

Like most practical problems, there is no single universal solution to the registration problem. As such, it is often helpful to further classify the problem based on the subject, imaging technique, and space of possible solutions. Registration is called *intra-subject* if the images belong to the same subject and *inter-subject* if the images are taken from different subjects. We call a registration problem *mono-modal* if the images are produced by the same imaging technique and *multi-modal* if the images are taken by different imaging techniques. Registration is called *multi-temporal* if the images depict structural changes of the same subject over time. For example, erosions of the earth's surface or tumor growth. Finally, we describe registration based on the class of transformations to which the solution belongs. In this context, a registration may be described as *rigid*, *similarity*, *affine*, *projective*, *polynomial*, or *deformable*. For example, when we call a registration, rigid, we imply that the solution is limited to the space of angle and distance preserving transformations. A formal description of different transformation types will be given in later sections.

Other classifications include *feature-based* and *content-based* registration, and *parametric* and *non-parametric* registration. In feature-based registration the so-

lution is derived by finding a transformation that optimally maps a finite set of features which has been identified for each image. Features can be manually selected or automatically detected, they can be derived from the subject such as corners and anatomical features or be artificially introduced by placing markers next to or on the subject.

In medical imaging, *fiducial markers* such as stereo-tactic frames can be attached to the patient during the imaging process and used to determine the alignment to a high accuracy. Despite their high accuracy, feature-based methods typically require some human input or intervention, which makes them undesirable for fully automated processes. On the other hand, content-based methods can be fully automated and be used retrospectively. Recent advances (in the last decade or so) in similarity measures such as the introduction of *mutual information* (MI) and improved optimization methods, have made the accuracy of content-based methods comparable to feature-based methods (e.g. see [1]) and the dominant class of image registration research. Unless otherwise noted, any reference to image registration in the rest of this thesis deals with the content-based registration.

For a classical and general treatment of the registration techniques, we refer the reader to [2]. For a survey of medical image registration in general refer to [3] and for mutual information-based registration of medical images refer to [4]. A modern treatment of the numerical methods for registration of medical images is provided in [5]. A survey of high performance medical image registration on multi-core, graphics processors and distributed architectures is given by the author in [6].

2.1 Problem Statement

We define an image \mathcal{I} as a mapping from \mathbb{R}^d to \mathbb{R} , $\mathcal{I} : \mathbb{R}^d \rightarrow \mathbb{R}$. Without loss of generality, we assume that images are defined such that $\mathcal{I} : [0, 1]^d \rightarrow [0, 1]$ and denote the space of all images by \mathbb{I} . For two images \mathcal{F} and $\mathcal{M} \in \mathbb{I}$, the registration problem is to find a mapping $T(\cdot)$ that minimizes some distance measure \mathcal{D} or maximizes a similarity measure \mathcal{S} defined on \mathbb{I} , or formally:

$$T_{opt} = \underset{T}{\operatorname{argmin}} \mathcal{D}(\mathcal{F}, \mathcal{M}(T)). \quad (2.1)$$

Image \mathcal{F} is known as the *fixed* or *reference* image and image \mathcal{M} as the *moving* or *template* image. The three main ingredients of the solution include:

- Choosing an appropriate class of transformations that can sufficiently describe the mapping between the images.

- A suitable distance measure which attains its minimum at the correct alignment between the two images, is sufficiently smooth with a sufficiently large attraction range, and with not so many local minima.
- A suitable optimization strategy which can find the global minimum despite local minima and within a reasonable time.

In the interest of making a general statement, we deliberately stay clear of defining qualitative terms such as sufficiently smooth, large attraction range and a reasonable time which need to be clarified in the context of each specific application.

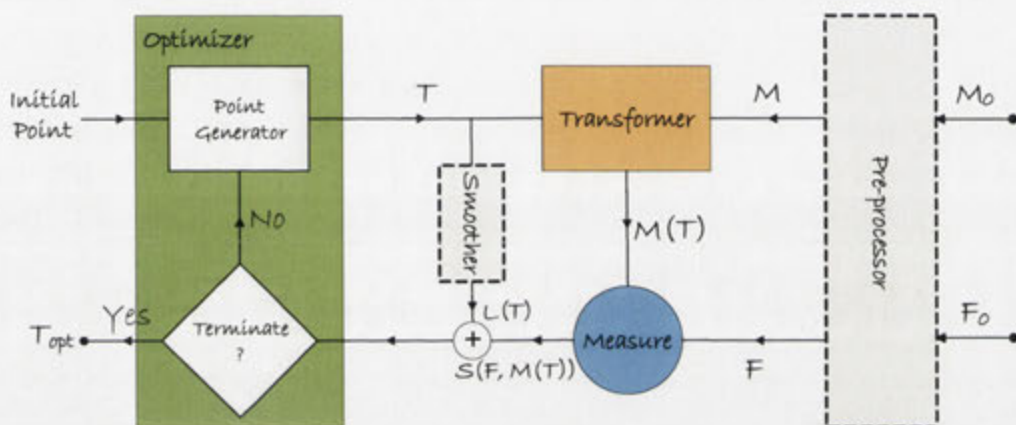


Figure 2.1: A general registration solver and its main components: F , M , and $M(T)$ are fixed, moving and transformed moving images, respectively.

Fig. 2.1 shows various components of a general registration solver, with the main components a *transformer*, a *measure*, and an *optimizer*, as described above. Registration as depicted here is an iterative process where the moving image is transformed within a predetermined parameter space and compared against the fixed image. A measure of similarity or distance is computed between the images at each step and used to determine if they are ‘sufficiently’ aligned. This process is controlled by the optimizer which starts from an initial guess and determines subsequent steps in order to reach an optimal alignment. We will discuss each component in more detail in the following sections.

2.2 Transformer

The transformer maps points in the moving image to new locations in the transformed image. Depending on the registration problem, a transformation can be *collinear* or *deformable*. Collinear transformations are line-preserving i.e. map

straight lines onto straight lines. Collinear transformations can be described by a 4×4 matrix acting on homogeneous vectors representing 3D points. Examples of collinear transformations include *rigid*, *similarity*, *affine*, and *projective*¹. For this reason, these types of transformations have nearly identical complexity. Methods that implement rigid registration can be easily extended to affine, often without any change to the transformer.

Deformable transformation methods can be further categorized as *parametric* and *non-parametric*. Non-parametric methods are based on a *variational* formulation of the registration problem, where the transformation is described by an arbitrary displacement field *regularized* by some smoothing criteria [5]. Parametric methods are based on some piecewise polynomial interpolation of a displacement field using a set of control points placed in the image domain. B-splines are commonly used in this context as they induce local deformations that limit the computational complexity of a large grid of control points [7]. Other functions such as thin-plate splines [8] and Bezier functions [9] have also been used. There are efficient methods for non-parametric registration including *multi-grid* solvers. While parametric methods are more demanding, they yield themselves more easily to multi-modal registration applications.

The transformer determines the intensity of the points in the transformed image by interpolating intensity values of corresponding points in the moving image. The simplest and fastest interpolation method is the *nearest neighbor* interpolation. Nearest neighbor should never be used in practice, as it results in poorly shaped cost functions, but may be useful to establish the baseline performance of the transformer. The most commonly used interpolation method is *linear interpolation*. Other methods include *quadratic*, *cubic*, *cubic B-spline*, and *Gaussian* interpolation [10].

A transformer spends the majority of its time performing interpolations. As noted by Castro-Pareja et al. [11], interpolation of the transformed moving image does not benefit from standard memory caching mechanisms due to non-sequential pattern of access to memory with low locality. As a result, transformer performance can well become memory-bound.

Parametric vs Non-Parametric

If transformation $T(\cdot)$ in (2.1) can be expressed in terms of a finite set of parameters, we call the registration parametric and non-parametric otherwise. Common types of parametric registration are briefly described in the following subsections. For a

¹Projective transformations are rarely required in medical imaging applications.

comprehensive and hierarchical treatment of transformations, the reader is referred to [12].

2.2.1 Rigid

A rigid transformation is used to recover the misalignment between the images when the mapping between the images is known to be angle and distance preserving. A rigid transformation consists of a translation and a proper rotation with 3 and 6 parameters in two and three dimensions, respectively.

A rigid transformation $T_r(\cdot)$ can be shown in *homogenous* block form as:

$$T_r(\mathbf{x}) = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \mathbf{x}, \quad (2.2)$$

where \mathbf{R} and \mathbf{t} are the rotation matrix and the translation vector, respectively.

2.2.2 Similarity

A similarity transformation is used when the mapping between the images can be described by a rigid transformation plus an isotropic scaling. The transformation preserves angle, relative distance and relative area. A similarity transformation can be specified by 4 and 7 parameters in two and three dimensions, respectively.

A similarity transformation $T_s(\cdot)$ can be shown in homogenous block form as:

$$T_s(\mathbf{x}) = \begin{bmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \mathbf{x}, \quad (2.3)$$

where s is the isotropic scaling.

2.2.3 Affine

An affine transformation is used when the mapping between the images can be described by a rigid transformation plus anisotropic scaling and sheer parameters. Parallel lines remain parallel under the transformation, so does relative distance on parallel lines and relative area. An affine transformation can be specified by 6 and 12 parameters in two and three dimensions, respectively.

An affine transformation $T_a(\cdot)$ can be shown in homogenous block form as:

$$T_a(\mathbf{x}) = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \mathbf{x}, \quad (2.4)$$

where A is an arbitrary 2×2 and 3×3 matrix for two- and three-dimensional transformations, respectively.

2.2.4 Projective

A projective transformation is used when the mapping between the images can be described by a linear transformation of homogenous coordinates. This is the most general type of a collinear transformation which preserves the *cross ratio* (ratio of ratios) of any four collinear points. A projective registration can be specified by 8 and 15 parameters in two and three dimensions.

A projective transformation $T_p(\cdot)$ can be shown in homogenous block form as:

$$T_p(\mathbf{x}) = \begin{bmatrix} A & \mathbf{t} \\ \mathbf{v}^T & v_0 \end{bmatrix} \mathbf{x}. \quad (2.5)$$

The transformation is defined up to a scale and as such, there are $N - 1$ degrees of freedom for a projective transformation with N elements.

2.2.5 B-Spline

Non-rigid deformation between images can be modeled by spline functions operating on a finite set of control points distributed across the image. In this section we look at a commonly used spline-based transformation which uses B-splines to model the deformation between the images [7].

Let Φ denote a mesh of $n_x \times n_y \times n_z$ control points $\phi_{ijk} = [x_{ijk}, y_{ijk}, z_{ijk}]^T$ distributed over an image and $\delta_x, \delta_y, \delta_z$ be the distance between adjacent control points if they were uniformly distributed. The cubic B-spline deformation in 3D can then be defined as

$$\mathbf{T}_b(x, y, z) = \sum_{\ell=0}^3 \sum_{m=0}^3 \sum_{n=0}^3 B_\ell(u) B_m(v) B_n(w) \phi_{i+\ell, j+m, k+n}, \quad (2.6)$$

where $B_\ell(\cdot)$ is the ℓ^{th} basis function of the cubic B-spline defined as

$$B_0(u) = \frac{(1-u)^3}{6}, \quad (2.7)$$

$$B_1(u) = \frac{3u^3 - 6u^2 + 4}{6}, \quad (2.8)$$

$$B_2(u) = \frac{-3u^3 + 3u^2 + 3u + 1}{6}, \quad (2.9)$$

$$B_3(u) = \frac{u^3}{6}, \quad (2.10)$$

and the i, j, k and u, v, w are calculated using

$$i = \lfloor \frac{x}{\delta_x} \rfloor - 1, \quad j = \lfloor \frac{y}{\delta_y} \rfloor - 1, \quad k = \lfloor \frac{z}{\delta_z} \rfloor - 1, \quad (2.11)$$

$$u = \frac{x}{\delta_x} - \lfloor \frac{x}{\delta_x} \rfloor, \quad v = \frac{y}{\delta_y} - \lfloor \frac{y}{\delta_y} \rfloor, \quad w = \frac{z}{\delta_z} - \lfloor \frac{z}{\delta_z} \rfloor. \quad (2.12)$$

One benefit of B-spline transformation is its computational efficiency compared to other spline-based transformations. The mapping of a given point is determined only by a mesh of 4×4 (for 2D transformations) or $4 \times 4 \times 4$ (for 3D transformations) of control points that are closest to the point being transformed. This means that changes to the position of the control points only affects points in their local neighborhood which is particularly useful in computation of the gradient measures using finite differences.

2.3 Similarity/Distance Measures

Just as different classes of transformations are suitable for modeling different geometric distortions between the images, different measures are used for different intensity distortions between the images. Measures are broadly categorized based on their suitability for single-modality and multi-modality problems. We discuss commonly used measures in this section a list of which is given in Table 2.1. We note that for certain measures it is more natural to talk about similarity rather than distance between the images. Since a similarity measure $\mathcal{S}(\cdot)$ is trivially related to a distance measure $\mathcal{D}(\cdot) = -\mathcal{S}(\cdot)$ and in line with the optimization literature, we will refer to minimizing a cost function even when we actually mean maximizing the corresponding similarity measure.

Typically, single-modality measures can be calculated by independent computations at each spatial location. From a parallelization point of view, this makes them readily adaptable to single instruction multiple data (SIMD) instruction sets

and architectures such as GPUs. Multi-modality measures determine statistical (mutual information) or functional (correlation ratio) dependance of images where each image is assumed to be a realization of an underlying discrete random variable. These methods require estimation of joint and marginal probability mass functions (pmfs) of the underlying discrete random variables from image data. Methods of pmf computation can be parallelized with varying degrees of difficulty and performance improvement. We will discuss this issue in more detail in the context of MI computation on the GPU in Section 4.2.3.

2.3.1 Sum of Squared Differences (SSD)

Sum of squared differences (SSD) is a direct measure of distance between two images. For $\mathcal{F}, \mathcal{M} \in \mathbb{I}$, SSD is defined as:

$$\mathcal{D}_{\text{SSD}}(\mathcal{F}, \mathcal{M}) = \sum_{\mathbf{x} \in \Omega} (\mathcal{F}(\mathbf{x}) - \mathcal{M}(\mathbf{x}))^2, \quad (2.13)$$

where $\Omega \subset \mathbb{R}^d$ represents the overlapping area between the two images. SSD is a computationally efficient measure for mono-modal registration. The assumption is that image intensities are (more or less) unchanged under the transformation. SSD can be shown to be an optimal distance measure where the image intensities only differ by a stationery Gaussian noise [13].

The main drawbacks of SSD are the underlying assumption that there is identity correspondence between the image intensities and its sensitivity to outliers. One approach to reduce sensitivity to outliers is to use the closely related distance measure, sum of absolute differences (SAD), defined as:

$$\mathcal{D}_{\text{SAD}}(\mathcal{F}, \mathcal{M}) = \sum_{\mathbf{x} \in \Omega} |\mathcal{F}(\mathbf{x}) - \mathcal{M}(\mathbf{x})|. \quad (2.14)$$

One problem with SAD is that its derivative can become singular, which makes it undesirable for optimization schemes that rely on the derivative of the distance measure, such as *gradient descent*-based methods.

2.3.2 Correlation Coefficient (CC)

Correlation coefficient² is an affine measure of similarity between two images. For $\mathcal{F}, \mathcal{M} \in \mathbf{I}$, CC is defined as:

$$\mathcal{S}_{CC}(\mathcal{F}, \mathcal{M}) = \sum_{\mathbf{x} \in \Omega} \frac{(\mathcal{F}(\mathbf{x}) - \mathbb{E}[\mathcal{F}(\mathbf{x})])(\mathcal{M}(\mathbf{x}) - \mathbb{E}[\mathcal{M}(\mathbf{x})])}{\sigma(\mathcal{F})\sigma(\mathcal{M})}, \quad (2.15)$$

where $\mathbb{E}[\cdot]$ and $\sigma(\cdot)$ are the expectation and standard deviation, respectively. CC can be viewed as the cosine of the angle between the two images which attains its maximum when image intensities are affinely dependent [5], i.e. if $\mathcal{M} = a\mathcal{F} + b$, then $\mathbb{E}[\mathcal{M}] = a\mathbb{E}[\mathcal{F}] + b$ and $\sigma(\mathcal{M}) = a\sigma(\mathcal{F})$ and hence $\mathcal{S}_{CC} = 1$.

CC is suitable for mono-modal registration particularly where there are brightness and contrast changes between the two images.

2.3.3 Mutual Information (MI)

Mutual information is a concept borrowed from the information theory and is based on the entropy of random variables. In this context, the image is assumed to be the realization of a discrete random process. Entropy of a random variable is a measure of the average or expected information content of an event, whose distribution is determined by the marginal probability of the random variable. One such measure was introduced by Shannon in 1948 [14], and is defined as

$$H(X) = \sum_{x \in X} p(x) \log \frac{1}{p(x)}, \quad (2.16)$$

where $p(\cdot)$ is the pmf³ of the random variable X . Shannon entropy measures the degree of uncertainty of a random variable by scoring less likely outcomes higher than the more likely ones. This is consistent with the notion that knowledge of an outcome that can be easily predicted is considered less valuable.

Mutual information of two random variables is the amount of information that each carries about the other and is defined as

$$\begin{aligned} I(X; Y) &= H(X) - H(X|Y) \\ &= H(X) + H(Y) - H(X, Y), \end{aligned} \quad (2.17)$$

²Some authors use normalized cross correlation (NCC) to refer to correlation coefficient. We prefer correlation coefficient which is the accepted term in statistics.

³We note that in the context of images we deal with discrete signals and discrete random variables. It is therefore more appropriate to use pmf than its continuous equivalent probability density function (pdf).

where $H(X|Y)$ is the information content of random variable X if Y is known, $H(X, Y)$ is the joint entropy of the two random variables and is a measure of combined information of the two random variables. $I(X; Y)$ can be thought of as the reduction in uncertainty of random variable X as a result of knowing Y . The uncertainty is maximally reduced, when there is a one-to-one mapping between the two random variables and is not reduced at all if the two random variables are independent and do not provide any information about one another.

In the context of images, MI is a statistical measure of similarity between two images. For $\mathcal{F}, \mathcal{M} \in \mathbf{I}$ and using (2.17), MI can be written as:

$$S_{\text{MI}}(\mathcal{F}, \mathcal{M}) = \sum_f \sum_m p_{\mathcal{F}\mathcal{M}}(f, m) \log \frac{p_{\mathcal{F}\mathcal{M}}(f, m)}{p_{\mathcal{F}}(f)p_{\mathcal{M}}(m)}, \quad (2.18)$$

where $p_{\mathcal{F}}(\cdot)$, $p_{\mathcal{M}}(\cdot)$, and $p_{\mathcal{F}\mathcal{M}}(\cdot)$ are the marginal and joint probability mass functions of discrete random variables \mathcal{F} and \mathcal{M} , respectively. MI as a similarity measure was first introduced by [15] and [16]. MI-based registration has since been used in numerous applications, particularly for multi-modal registration of medical images (for a comprehensive list refer to [4]).

A commonly used alternative to MI is normalized mutual information (NMI) defined as:

$$S_{\text{NMI}}(\mathcal{F}, \mathcal{M}) = \frac{2S_{\text{MI}}(\mathcal{F}, \mathcal{M})}{H(\mathcal{F}) + H(\mathcal{M})}. \quad (2.19)$$

There is anecdotal evidence that NMI is less sensitive to partial overlap between the images [17, 18]

We also note that while, in theory, maximization of MI is equivalent to minimization of the joint entropy, the latter is hardly used in practice. This is due to the fact that joint entropy has a tendency to drive the optimization towards solutions where images have little or no overlap. This is where joint entropy itself is degenerate and close to zero. MI avoids this pitfall since as the overlap between the images gets smaller so does the information content of each image and consequently the mutual information itself is reduced and gets farther from a local maximum.

2.3.4 Correlation Ratio (CR)

Correlation ratio (CR) measures a functional correlation between the images without making an explicit assumption regarding the nature of the function itself. Development of CR was apparently motivated by the observation that MI was too

under-constrained for certain applications [19], as it does not explicitly take the spatial relationship between the image intensities into account.

Correlation ratio for two random variables X and Y is defined as

$$\eta(X; Y) = \frac{\sigma^2(\mathbb{E}[Y|X])}{\sigma^2(Y)}. \quad (2.20)$$

CR is defined in the $[0,1]$ range and measures the functional dependence between the two random variables. It assumes a value of 0 when there is no functional dependence and 1 when there is a purely deterministic dependence between the two random variables. We note that CR is in general non-symmetrical [19].

Both MI and CR require estimation of the joint pmf of the images and as such their complexity is $\mathcal{O}(n^2)$ with n being the number of intensity levels or bins for each image. However if the pmfs are simply calculated by normalization of the joint histogram of the images, an $\mathcal{O}(n)$ implementation of CR is possible [20].

CR fills the gap between CC and MI, where it is not as constrained and as restricted as CC nor as unconstrained and spatially ignorant as MI. Hence, the measure is useful when images are known or expected to have some functional relationship.

2.3.5 Probability Distribution Estimation

One often overlooked aspect of image registration with MI or CR as the similarity measure is the need for a robust estimation of the joint and marginal probability mass functions (pmf) from image content. Most researchers (as surveyed by [4]) use a standard joint histogram of intensity co-occurrences as an estimation of the pmf. However, as we will demonstrate in Section 3.1.6, this method is not robust w.r.t. the number of histogram bins. The choice of number of bins affects the similarity measure and in turn the optimization and registration accuracy. A smaller than optimal number of bins typically results in an over-smoothed cost function and a less accurate registration while a larger than optimal number of bins results in a cost function with lots of ripples and can result in mis-registration.

An efficient method to solve this problem is to use the *uniform volume histogram* (UVH) [21] for estimating the pmfs. We will discuss UVH in Section 3.1.6.

2.3.6 Comparison of Measures

Table 2.1 shows a quick comparison of the measures discussed in this section. The measures are listed from the least general (SSD) to the most general (MI) but the

computational complexity increases from top to bottom.

Table 2.1: Comparison of Similarity/Distance Measures

Measure	Type	Mapping	Pros	Cons
SSD	Distance	One-to-one	<ul style="list-style-type: none"> • Suitable for mono-modal registration • Suitable for non-parametric registration • Computationally efficient • Easy to differentiate 	<ul style="list-style-type: none"> • Limited to mono-modal registration • Sensitive to outliers • Sensitive to non-overlapping areas
SAD	Distance	One-to-one	<ul style="list-style-type: none"> • Suitable for mono-modal registration • Computationally efficient • Less sensitivity to outliers compared to SSD 	<ul style="list-style-type: none"> • Limited to mono-modal registration • Differentiation introduces singularity
CC	Similarity	Affine	<ul style="list-style-type: none"> • Suitable for mono-modal registration • Not sensitive to brightness and contrast changes 	<ul style="list-style-type: none"> • Limited to mono-modal registration
CR	Similarity	Functional	<ul style="list-style-type: none"> • Suitable for multi-modal registration • Only requires an implicit functional relationship between the images 	<ul style="list-style-type: none"> • Computationally expensive
MI	Similarity	Statistical	<ul style="list-style-type: none"> • Suitable for multi-modal registration • Least sensitivity to outliers • Requires no explicit or implicit relationship between the images 	<ul style="list-style-type: none"> • Computationally expensive

2.4 Optimizer

Image registration typically involves an optimization step, where the transformation $T(\cdot)$ is iteratively refined from a initial guess $T_0(\cdot)$ until some convergence criterion is met. The optimizer is responsible for an efficient and often non-exhaustive strategy to search the transformation parameter space for the best match between the images. In image registration, optimizers can be broadly categorized as *gradient-based* or *gradient-free*, *global* or *local*, and *serial* or *parallelizable*.

Gradient-based methods require computation of partial derivatives of a cost function in addition to frequent computation of the cost function itself. Therefore, from an implementation perspective, they are more involved than gradient-free methods. The gradient computation can be based on the numerical estimation of the derivatives using finite differences. Alternatively, direct computation of the gradient can be performed when closed-form equations for the partial derivatives can be derived.

Local methods find a local optimum in the vicinity of an initial point and within their *capture range*. They may converge to an incorrect alignment if not properly

initialized. The problem can be alleviated by using an appropriate distance measure with a single dominant minima, limiting the dynamic range of the parameters and initializing the optimization sufficiently close to the global minimum and within the capture range of the distance measure to ensure the desired correct convergence. In practice, all the above strategies need to be employed to guarantee the success of a registration algorithm. Global methods, however, find the global optimum within a given range of parameters. They are robust with respect to selection of the initial point but at the cost of slower convergence. Global and local methods may be combined to improve robustness while maintaining a reasonable convergence rate. For parametric registration, it is often the case that a global optimization of the parameter space is computationally prohibitive and one needs to resort to a local optimization method.

Some optimization algorithms are only suited for serial execution, where each optimization step is dependent on the outcome of previous step(s). Others may be amenable to parallelization. For example, each step of the gradient descent optimization in N -dimensional space requires computation of N partial derivatives of the cost function. Here, there is limited opportunity to run up to N tasks in parallel, and of course the additional line minimization step that may follow cannot be readily parallelized. We call such methods partially parallelizable. And finally, we refer to optimization methods that have been designed for parallel execution with minimal inter-step dependency as fully parallelizable.

Table 2.2 lists some optimization algorithms used for image registration and their respective classification.

The overall performance of a registration algorithm is dependent on the effectiveness of the optimization strategy. This in turn depends on the *iterations* needed for the algorithm to converge. For gradient-free optimization, we define an iteration as a step which involves a single computation of the cost function. For gradient-based optimization, an iteration is defined as a step that involves a single computation of the gradient. Depending on the type of gradient-based method this may involve several evaluations of the cost function.

Gradient-based optimizers do more in a single iteration and they also converge with a significantly lower number of iterations compared to gradient-free methods. The convergence rate of an optimizer depends on many factors including the size of the parameter space, optimizer settings (e.g. convergence criteria), and the misalignment between the images. It is also often data-dependent.

A sample breakdown of computations in one iteration of a gradient-free optimization algorithm is given in Table 2.3 for affine registrations using a single

Table 2.2: Classification of commonly used optimization methods

Method		Classification	
Powell [22]	gradient-free	local	serial
Simplex [23]	gradient-free	local	partially parallelizable
Soblex ¹ [21]	gradient-free	combined	partially parallelizable
MDS ^{1,2} [24]	gradient-free	local	partially parallelizable
Gradient descent [22]	gradient-based	local	partially parallelizable
Quasi-Newton [22]	gradient-based	local	partially parallelizable
Levenberg-Marquardt [22]	gradient-based	local	partially parallelizable
Simulated annealing [22]	gradient-free	combined	partially parallelizable
DIRECT ³ [25]	gradient-free	global	fully parallelizable
Genetic [26]	gradient-free	global	fully parallelizable

¹ : a simplex variant, ² : multidirectional search, ³ : dividing rectangles

modality and a multi-modality measure. The measurements are based on a Quad core Intel Core i7 920 and an NVIDIA GTX 295. The time spent outside of the measure and transformation components is negligible compared to the measure and transformation. On the CPU the execution time is dominated by the transformer whereas on the GPU, the time spent in computing the measure, particularly for MI, exceeds the transformer time. This is expected as GPUs are designed to speed up geometric transformations. Obviously, for more complex transformation models such as the deformable B-splines more time will be spent in the transformer for both platforms.

Table 2.3: A sample breakdown of computations for affine registrations on a multi-core CPU and a GPU

	Affine (SSD)			Affine (MI)		
	Measure	Transform	Other	Measure	Transform	Other
CPU	4.3%	95.7%	< 0.1%	13.5%	86.5%	< 0.1%
GPU	50.4%	49.2%	0.4%	86.9%	13.0%	0.1

As can be seen in Table 2.3, the computational bottleneck of registration is not the optimizer but the computation of the transformation and the measure. For this reason, most researchers focused reducing the computational cost of these components such as their *fine-grained* parallelization. A few have considered *coarse-grained* parallelization which involves parallelization of the optimizer itself [27,28].

In the following subsections we will briefly describe common local optimization methods. The reader is referred to [29] for a more comprehensive treatment.

2.4.1 Powell

Powell's multi-dimensional direction set algorithm finds the minimum of a function by iteratively minimizing the function along a set of N directions, where N is the number of independent parameters of the cost function. At its core, the method uses a line minimization method (typically Brent's method [30]). In the absence of any direction, the parameter space's origin is commonly used as the starting position for the optimization algorithm.

One problem with the Powell algorithm is that from an observer's point of view who knows where the minimum is located, it appears to spend a lot of time, aimlessly iterating on and around the minimum. Obviously, the only way the optimization algorithm can satisfy itself that it has found the actual minimum is to spend enough time to check the perimeter [31].

If the algorithm can be initialized close to the minimum, one can use the knowledge to speed up convergence by refraining from checking the perimeter excessively as we have described in [31]. The Powell algorithm can be modified to keep track of each point in the N -dimensional space that it evaluates. A minimum distance or resolution is defined and the cost function is only evaluated when a new point falls outside all previously evaluated points by the specified minimum distance. The cost function quickly returns a previously calculated value for points inside the minimum distance of a previously evaluated point, which limits the resolution of the cost function. Obviously, this method is not suitable as a general purpose optimization tactic, and can only be used when the optimization algorithm can be properly initiated.

In our experience, Powell produces accurate results in registration applications when initialized close to the global minimum, however it has slower convergence rate and is more sensitive to local minima compared to Simplex optimization. Powell does not require derivatives of the cost function which makes it an attractive solution where the derivative of the cost function is difficult to calculate or may become singular.

2.4.2 Simplex

A *simplex* is a geometrical object that consists of $N + 1$ vertices in N dimensions. 2D and 3D simplexes are known by the more familiar names of triangle and tetra-

hedron, respectively. The algorithm is named Simplex because at each iteration, it evaluates the cost function at $N + 1$ vertices of a simplex for an N -parameter problem. The algorithm keeps track of the best (lowest cost), second best and the worst (highest cost) vertices and attempts to climb downhill through a number of basic moves as described below.

Reflection: the worst vertex is reflected through the opposite face. Reflection is the most common move in Simplex and allows for the algorithm to quickly climb downhill.

Expansion: if a reflection results in the best vertex to be replaced by the new point, an expansion is attempted to see if the cost can be further improved along the reflected direction.

Contraction: if the reflected vertex is not as good as the best or the second best vertices, the worst vertex is moved closer to the opposite face. A contraction typically occurs when the simplex is settling at the bottom of a valley and moving in the direction of a reflection may take the algorithm uphill.

Full Contraction: if none of the previous strategies work, all vertices except for the best vertex are moved closer to the best vertex along the simplex sides.

The algorithm is repeated until some convergence criterion is met.

Similar to the Powell method, Simplex only requires evaluation of the function itself and not the derivatives.

2.4.3 Soblex

If a cost function contains several local minima and a single dominant minimum, local optimization method can be easily trapped by the local minima. While not desirable, this may be acceptable for certain applications, where one only needs to improve on an initial cost. However, in registration applications, being trapped by a local minimum results in mis-registration and complete failure of the algorithm. Soblex [21] is a quasi-global optimization method based on simplex and sampling of the parameter space with the Sobol quasi-random sequence [32].

The Soblex optimization is initially given a budget, in terms of time or number of cost function calls. Within the initial budget, Soblex evaluates the cost function using the Sobol sequence and initializes a simplex-shaped subspace, which is constructed from points with the lowest costs. The Sobol sequence ensures that we can progressively sample the parameter space in a virtually uniform fashion. Intuitively, if the budget is large enough, the simplex subspace can sufficiently close in on the global minimum to allow successful execution of the optimization algorithm [21].

Unlike most optimization methods that start with a single point in space, the simplex algorithm allows us to start from a region of space that can arbitrarily be made close to the global minimum by increasing the Soblex budget. Quasi-random sampling of the parameter space is preferred over a uniform sampling, as it avoids sample-set's bias as a result of a uniform distance between the samples which is also known as the *grid-effect*. A Sobol sequence is also preferred over a true random distribution as it guarantees that the each subspace is given a more-or-less equal weight regardless of the number of samples drawn.

Soblex optimization is one of the contributions of this thesis and has been used for optimization of rotational parameters in *Gradient-Intensity*-based registration where Powell and Simplex would fail [21].

2.4.4 Gradient Descent

Gradient descent is a popular gradient-based optimization algorithm, largely due to its simple structure and ease of implementation. Once the gradient is computed, the choices include taking a single step in a direction opposite to the gradient where the step size may be adjusted over time, or use of a line minimization algorithm such as Brent's [22]. Line minimization usually involves several computations of the cost function alone without its derivatives.

When comparing results it is important to identify which variation of the gradient descent is used. We have come across four different implementations.

- Type A: Closed-form differentiation with a single step
- Type B: Closed-form differentiation with line minimization
- Type C: Numerical differentiation with a single step
- Type D: Numerical differentiation with line minimization.

Gradient descent is, however, known to be slow to converge compared to other gradient-based optimization methods such as *conjugate gradient* or *quasi-Newton* methods [22].

2.4.5 Functional Optimization Using Variational Calculus

For the sake of completeness, we discuss functional optimization which is used in non-parametric registrations. Consider the following minimization problem, where

we would like to find a function $u_{\text{opt}}(\cdot)$ which minimizes the following expression:

$$u_{\text{opt}} = \underset{u}{\operatorname{argmin}} \underbrace{\int_{x_1}^{x_2} F(x, u(x), u'(x)) dx}_{\psi}, \quad (2.21)$$

subject to some boundary condition $u(x_1) = \alpha$, $u(x_2) = \beta$. Here, $F(\cdot)$ is a function of the independent variable x , the unknown function $u(\cdot)$ and its derivative and is called a *functional*. It turns out that many practical problems including deformable registration can be formulated as (2.21) and the standard method to solve them is to solve the partial differential equation arising from the following Euler-Lagrange equation:

$$F_u - \frac{d}{dx} F_{u'} = 0, \quad (2.22)$$

where F_u and $F_{u'}$ are derivatives of $F(\cdot)$ w.r.t. u and u' , respectively.

Proof

For function $u(\cdot)$ to be the solution to the minimization problem in (2.21), we expect the *Gâteaux* derivative of ψ in the direction of an arbitrary function $\eta(\cdot)$ be zero:

$$d\psi(u; \eta) = \lim_{\varepsilon \rightarrow 0} \frac{\psi(u + \varepsilon\eta) - \psi(u)}{\varepsilon} = 0, \quad (2.23)$$

replacing $u + \varepsilon\eta$ in (2.21) and using the Taylor series expansion of $F(\cdot)$ we have

$$\lim_{\varepsilon \rightarrow 0} \frac{\int_{x_1}^{x_2} [\varepsilon\eta(x)F_u(x, u(x), u'(x)) + \varepsilon\eta'(x)F_{u'}(x, u(x), u'(x)) + \varepsilon^2\zeta] dx}{\varepsilon} = 0, \quad (2.24)$$

where $\varepsilon^2\zeta$ represents higher order terms in the Taylor expansion. Taking the limit with $\varepsilon \rightarrow 0$ we have

$$\int_{x_1}^{x_2} \eta(x)F_u dx + \int_{x_1}^{x_2} \eta'(x)F_{u'} dx = 0. \quad (2.25)$$

We now use integration by parts on the second term

$$\int_{x_1}^{x_2} \eta(x)F_u dx + [\eta(x)F_{u'}]_{x_1}^{x_2} - \int_{x_1}^{x_2} \eta(x)\frac{dF_{u'}}{dx} dx = 0. \quad (2.26)$$

To satisfy the boundary condition we require $\eta(x_1) = \eta(x_2) = 0$

$$\int_{x_1}^{x_2} \eta(x) \left[F_u - \frac{dF_{u'}}{dx} \right] dx = 0. \quad (2.27)$$

The previous equation needs to hold for an arbitrary function $\eta(x)$, hence

$$F_u - \frac{d}{dx} F_{u'} = 0. \quad \square \quad (2.28)$$

In a similar manner, one could extend the result to a function of multiple variables and a vector field. The Euler-Lagrange equation for a function $u(\cdot)$ of d variables can be written as:

$$F_u - \sum_{i=1}^d \frac{\partial}{\partial x_i} F_{u_{x_i}} = 0, \quad (2.29)$$

where u_{x_i} is the partial derivative of $u(\cdot)$ w.r.t. the i^{th} variable x_i . The general form of Euler-Lagrange equations for a vector field $\mathbf{u}(\cdot) = [u_1(\cdot) \ u_2(\cdot) \ \cdots \ u_m(\cdot)]^T$ of d variables can be written as:

$$\mathbf{F}_u - \left(\frac{\partial}{\partial \mathbf{x}}^T \mathbf{F}_{\mathbf{u}_x} \right)^T = 0, \quad (2.30)$$

where

$$\mathbf{F}_u = \begin{bmatrix} F_{u_1} \\ F_{u_2} \\ \vdots \\ F_{u_m} \end{bmatrix}_{m \times 1}, \quad \frac{\partial}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial}{\partial x_1} \\ \frac{\partial}{\partial x_2} \\ \vdots \\ \frac{\partial}{\partial x_d} \end{bmatrix}_{d \times 1}, \quad \mathbf{F}_{\mathbf{u}_x} = \begin{bmatrix} F_{u_1 x_1} & F_{u_2 x_1} & \cdots & F_{u_m x_1} \\ F_{u_1 x_2} & F_{u_2 x_2} & \cdots & F_{u_m x_2} \\ \vdots & \vdots & \ddots & \vdots \\ F_{u_1 x_d} & F_{u_2 x_d} & \cdots & F_{u_m x_d} \end{bmatrix}_{d \times m}. \quad (2.31)$$

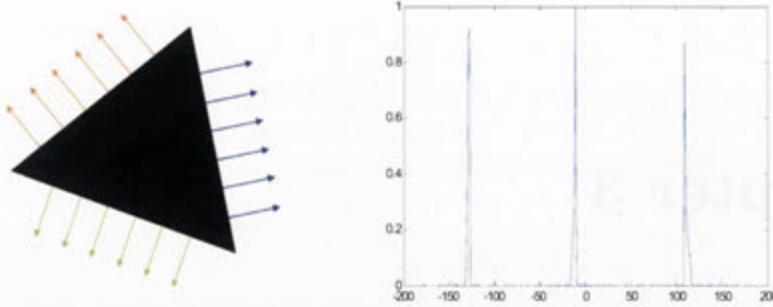
For a discussion of constrained problems and some examples refer to [33].

Chapter 3

Gradient Intensity-Based Registration

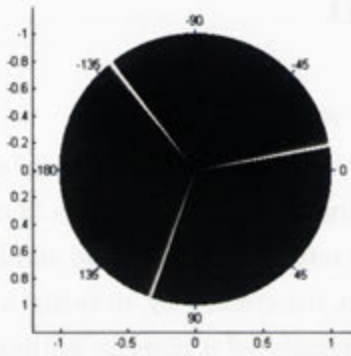
In this chapter, we introduce the concept of *gradient intensity* (GI), a measure of directional strength of an image based on its gradient content. We also discuss the applications of gradient intensity in rigid and similarity registration of images. Let us describe the concept informally by drawing a parallel with the intensity of a pixel in the image. Intensity of a pixel in an image represents strength of a signal measured at a given point in space. For images taken by a charge-coupled device (CCD) array camera, for instance, the pixel intensities are proportional to the amount of light collected by a CCD element in a given position of the array. For images taken by a CT scanner, the voxel intensities are proportional to the attenuation of an X-ray signal at a given position in space. Similarly, gradient intensity measures the number of significant gradient vectors in a gradient image that point to a given direction. In its simplest form, GI is a normalized histogram of gradient directions of the gradient field of an image.

Fig. 3.1 shows an image of a triangle and its main gradient directions. Excluding gradient vectors at (and close to) the vertices, there are three gradient directions each corresponding with one edge of the triangle. Fig. 3.1(b) shows the gradient intensity plot for different orientations. In theory, there should be only three gradient directions but in practice due to image quantization and numerical estimation of gradients other directions are present in addition to the main ones. We can also map gradient intensities to the circumference of a unit circle or on the surface of a unit disk as shown in Fig. 3.1(c), where brighter colors indicate higher gradient intensities in a given direction. For 2D images, GI is a mapping from a 2D vector field to a 1D scalar field. Similarly for 3D images, GI is a mapping from a 3D vector field to a 2D scalar field. A detailed discussion of GI for 2D and 3D images is given



(a) An image of a triangle: three main gradient directions shown

(b) Histogram of gradient orientations



(c) Gradient intensity of the triangle mapped on a unit disk

Figure 3.1: Gradient intensity of a triangle. The positive direction on the disk is clockwise. This aligns gradient intensity on the disk with the direction of gradients in the image for a better visual comparison.

in Section 3.1.2 and Section 3.1.3, respectively. An alternative interpretation of the GI based on a Hough transform is given in Section 3.1.5.

Gradient Intensity-Based Registration - The General Idea

There are two major limitations with automatic registration methods: (a) the robustness of the optimization is dependent on the shape of the cost function and can often lead to mis-registration due to presence of local minima, (b) the computation of the cost function involves image transformation and the computation of the similarity measure and needs to be performed several times which can be time consuming.

In MI-based registration the local minima problem is further exacerbated due to weak spatial constraining in the computation of MI. There are two broad ap-

proaches to solving the local minima problem for MI-based registration of images: (a) reducing local minima by improving the shape of the similarity function (b) skipping local minima by starting the optimization step closer to the actual alignment.

Pluim et al. [34] introduce spatial constraints to an MI-based cost function using a correction factor based on co-occurrence of gradient vectors. Liu et al. [35] propose an adaptive combination of intensity MI and gradient field MI with a multi-resolution approach to improve the shape of the cost function. The improved performance comes at the cost of increasing the computational complexity of the cost function calculation, which requires time consuming gradient image computation in addition to standard transformation and MI computation at each iteration of the optimization algorithm. Both methods have shown improved performance over NMI particularly for lower-resolution and down-sampled images. Despite favorable results, it cannot be analytically established whether the combined measures will always improve the smoothness of the cost function and can sufficiently cancel one-another's local minima and prevent formation of further ripples. Moreover, selection of coefficients, used in combined measures, is non-trivial and may well depend on the domain of the registration problem.

In this chapter, we look at improving the robustness and efficiency of image registration for the class of angle-preserving transformations (rigid and similarity) using the gradient intensity of images. We use the second approach in dealing with local minima by initializing the optimization close to the global minimum while reducing a computational complexity at the same time. The main idea is to break down a registration problem into simpler problems in multiple stages by decoupling the optimization for transformation parameters. To that end, we first obtain an initial estimate of the rotation parameter(s). With images rotationally aligned, we then look at efficient methods to estimate translation and scale parameters. Finally, the estimates of transformation parameters can then be used to initialize a suitable pixel intensity (PI)-based optimization algorithm. Since, the PI-based algorithm is initialized close to the actual optimum, it is guaranteed to converge to the correct alignment. As such, the overall algorithm has an extended capture range compared to a registration method that solely relies of pixel intensities and is quicker to converge due to being initialized closed to the optimum. The overall method's accuracy is equivalent to the PI-based method's accuracy.

Table 3 lists and compares corresponding concepts in GI- and PI-based registration methods.

We describe our method starting with simpler 2D registrations and registration

Table 3.1: Comparison of corresponding pixel- and gradient-based concepts.

PI-Based Method	GI-Based Method
Pixel	Direction
Intensity of a pixel	Number of gradient vectors in a direction
Intensity of a pixel assumed to be a random variable	Number of gradient vectors in each direction assumed to be a random variable
MI measures similarity of pixel intensities	MI measures directional similarity of gradient vectors
MI computed the entire image data	MI computed over a fixed-size histogram
Dimensionality of the problem is d	Dimensionality of the problem is $d - 1$

in Hough space to demonstrate the main ideas in Section 3.2 and Section 3.4 and then generalize to registration of 3D multi-modal images in Section 3.5. But first, let us take a more in depth look at the concept of the gradient intensity.

3.1 Gradient Intensity - The Concept

3.1.1 Gradient Field of an Image

The gradient field of a d -dimensional image such as $\mathcal{I}(x_1, x_2, \dots, x_d)$ is given by

$$\nabla \mathcal{I} = \left[\frac{\partial \mathcal{I}}{\partial x_1} \quad \frac{\partial \mathcal{I}}{\partial x_2} \quad \dots \quad \frac{\partial \mathcal{I}}{\partial x_d} \right]^T. \quad (3.1)$$

To compute the gradient for discrete images, the differentiation is replaced with a numerical estimation of the derivative such as finite differences. More generally, the differentiation is performed by a convolution with a suitable kernel in each direction:

$$\nabla \mathcal{I} = [\mathbf{K}_1 \otimes \mathcal{I} \quad \mathbf{K}_2 \otimes \mathcal{I} \quad \dots \quad \mathbf{K}_d \otimes \mathcal{I}]^T. \quad (3.2)$$

Common kernels include central differences, Sobel [36], and (truncated) Gaussian kernels. 2D versions of these common kernels are given in Table 3.2 where

Table 3.2: Examples of common 2D differentiating kernels

Kernel	K_x	K_y
Central differences	$\frac{1}{2} \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$	$\frac{1}{2} \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$
Sobel	$\frac{1}{8} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$	$\frac{1}{8} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$
Gaussian ¹	$\begin{bmatrix} 0.1370 & 0 & -0.1370 \\ 0.2259 & 0 & -0.2259 \\ 0.1370 & 0 & -0.1370 \end{bmatrix}$	$\begin{bmatrix} 0.1370 & 0.2259 & 0.1370 \\ 0 & 0 & 0 \\ -0.1370 & -0.2259 & 0.1370 \end{bmatrix}$

¹ : 3×3 truncated Gaussian $\sigma = 1$, $\mu = 0$

image coordinates with x -axis from left to right and y -axis from top to bottom is assumed.

The resulting vector field can be expressed in polar coordinates for 2D images and in spherical coordinates for 3D images. For 3D images the spherical representation at spatial location \mathbf{x} is given by $\mathbf{g}(\mathbf{x}) = [\rho_g(\mathbf{x}) \ \phi_g(\mathbf{x}) \ \theta_g(\mathbf{x})]^T$, where ρ_g , ϕ_g and θ_g represent magnitude, zenith and azimuth angles, respectively.

$$\rho_g(\mathbf{x}) = \sqrt{\mathcal{I}_x^2(\mathbf{x}) + \mathcal{I}_y^2(\mathbf{x}) + \mathcal{I}_z^2(\mathbf{x})}, \quad (3.3)$$

$$\theta_g(\mathbf{x}) = \arctan \frac{\mathcal{I}_y(\mathbf{x})}{\mathcal{I}_x(\mathbf{x})}, \quad -\pi \leq \theta_g(\mathbf{x}) < \pi \quad (3.4)$$

$$\phi_g(\mathbf{x}) = \arccos \frac{\mathcal{I}_z(\mathbf{x})}{\rho_g(\mathbf{x})}, \quad -\frac{\pi}{2} \leq \phi_g(\mathbf{x}) \leq \frac{\pi}{2}. \quad (3.5)$$

The polar representation of 2D gradient vectors $\mathbf{g}(\mathbf{x}) = [\rho_g(\mathbf{x}) \ \theta_g(\mathbf{x})]^T$ can be obtained by using (3.3) and (3.4) and setting $\mathcal{I}_z(\mathbf{x}) = 0$.

Gradient angles are defined everywhere except where $\rho_g(\mathbf{x}) = 0$. In practice, gradients whose magnitude is close to zero are sensitive to noise, quantization and interpolation errors. GI matrices, as described in the next section, are based on

gradient angles, so we would like to remove gradient vectors whose angles cannot be relied upon. For this purpose a binary gradient map function f_m is define as

$$f_m(\mathbf{x}) = \begin{cases} 1, & \rho_g(\mathbf{x}) > t \\ 0, & \rho_g(\mathbf{x}) \leq t \end{cases}, \quad \mathbf{x} \triangleq [x \ y \ z]^T, \quad (3.6)$$

where t is the gradient magnitude threshold.

Conventionally, an empirical or a statistical value such as the root mean square (RMS) is used as the threshold. However lack of a principled method for selecting t may either result in the removal of important gradient vectors or may leave too many sensitive vectors behind. To resolve this problem, we use a noise-resistant phase histogram method, described in 3.1.6, which removes sensitivity to the gradient threshold and allows us to conveniently set $t = 0$.

3.1.2 Gradient Intensity for 2D Images

For 2D images, gradient intensity is defined as the number of significant gradient vectors which pass through a given segment on the circumference of the unit circle.

We define the gradient intensity at the center of the segment specified by the angular resolution $2\delta\theta$

$$\Gamma(\theta) = \int \int h_\theta(\mathbf{x}) f_m(\mathbf{x}) dx dy, \quad (3.7)$$

where Γ is the gradient intensity at the center of the bin specified by θ and $h_\theta(\mathbf{x})$ is defined as

$$h_\theta(\mathbf{x}) = \begin{cases} 1, & \theta - \delta\theta \leq \theta_g(\mathbf{x}) < \theta + \delta\theta \\ 0, & \text{elsewhere} \end{cases}. \quad (3.8)$$

In order to be able to compare gradient intensity of different images, we use a normalized version of (3.7)

$$\tilde{\Gamma}(\theta) = \frac{\Gamma(\theta)}{\max \Gamma(\theta)}. \quad (3.9)$$

Gradient intensity as defined above gives a measure of the directional strength or content of an image.

Fig. 3.2 shows a number of images and corresponding gradient intensities encoded on a unit disk. Brighter colors indicate higher gradient intensities in a given direction. As can be seen, the gradient intensity disks clearly identify the rota-

tion between the images. However, a one-to-one relationship between the GI maps does not exist. For example a strong response is observed at $\frac{k\pi}{2}$ directions due to inherent bias of rectangular gradient kernels to these directions. In the case of Fig. 3.2(g) and Fig. 3.2(h), the effect of multi-modal nature of the images can be seen in GI maps, where the gradient intensities cannot be linearly related. These observations indicate that a statistical similarity measure such as MI that does not assume a linear relationship between gradient intensities will be more suitable for finding the correct mapping between the GI of the images.

3.1.3 Gradient Intensity for 3D Images

Gradient intensity is a measure of directional strength of an image. For 3D images, it is defined as the number of significant gradient vectors which pass through a given area on the surface of the unit sphere. Fig. 3.3 depicts gradient intensity mapped on the surface of the unit sphere with sample gradient vectors.

We define the gradient intensity at the center of the area specified by the angular resolution $2\delta\phi$ and $2\delta\theta$

$$\Gamma(\phi, \theta) = \int \int \int h_\phi(\mathbf{x}) h_\theta(\mathbf{x}) f_m(\mathbf{x}) dx dy dz, \quad (3.10)$$

where Γ is the gradient intensity at the center of the bin specified by (ϕ, θ) and $h_\phi(\mathbf{x})$ and $h_\theta(\mathbf{x})$ are defined as

$$h_\phi(\mathbf{x}) = \begin{cases} 1, & \phi - \delta\phi \leq \phi_g(\mathbf{x}) < \phi + \delta\phi \\ 0, & \text{elsewhere} \end{cases}, \quad (3.11)$$

$$h_\theta(\mathbf{x}) = \begin{cases} 1, & \theta - \delta\theta \leq \theta_g(\mathbf{x}) < \theta + \delta\theta \\ 0, & \text{elsewhere} \end{cases}. \quad (3.12)$$

To compare gradient intensity of different volumes we use a normalized version of (3.10)

$$\tilde{\Gamma}(\phi, \theta) = \frac{\Gamma(\phi, \theta)}{\max \Gamma(\phi, \theta)}. \quad (3.13)$$

Gradient intensity is different from a Gaussian sphere [37] representation of the orientation. A Gaussian sphere representation only captures the orientation of the surface normals, ignoring information inside the object (e.g. internal brain organs). It also assumes that there is an object to start with, hence implying that the object of interest is already segmented. The gradient intensity does not require

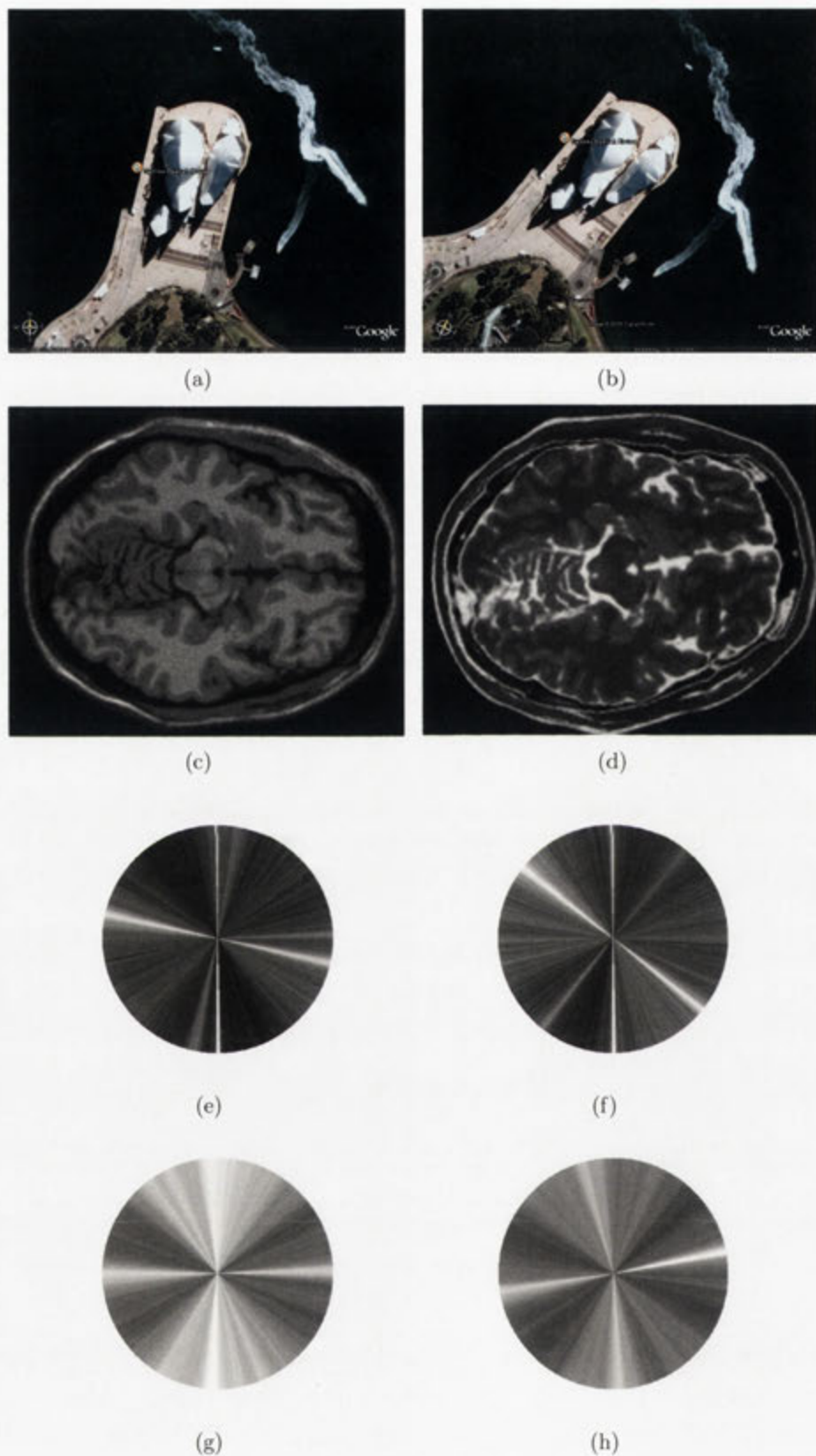


Figure 3.2: (a,b) Satellite images of Sydney Opera House, misaligned by -26.3° . (c,d) MR-T1 and MR-T2 images of brain, misaligned by a transversal rotation of 10° (e-h) Corresponding gradient intensities mapped on unit disks.

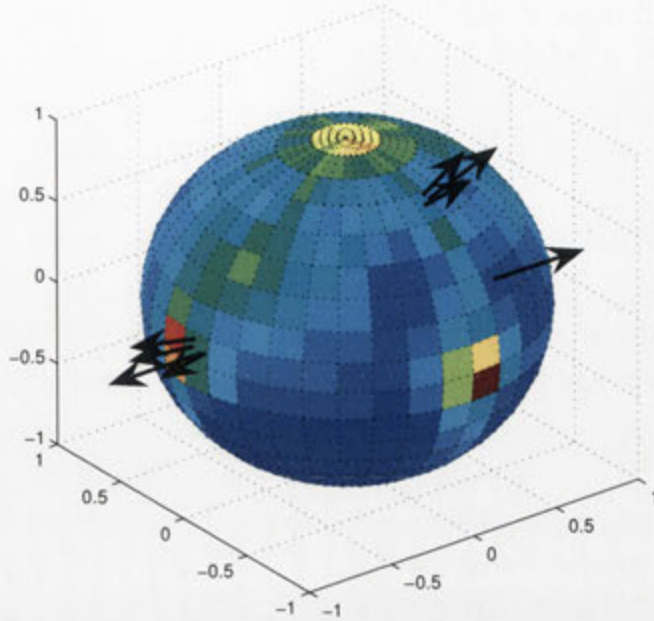


Figure 3.3: Gradient intensity mapped on the surface of the unit sphere. Intensity of each cell is a function of the number of gradients that pass through the cell. Sample gradient vectors are displayed for three cells.

segmentation and utilizes all the information which is in a 3D medical image and as such, is more suitable for the registration problem.

3.1.4 Entropy and MI of Gradient Intensity

Entropy of gradient intensity is a measure of directional information content of an image. The more directionally versatile an image, the higher the entropy. For example, in polygons, the entropy of gradient intensity is higher for higher order polygons with the maximum entropy observed for a circle and the minimum for a line.

Mutual information of gradient intensity of two images is a measure of directional similarity of images regardless of their relative size (scale) and position (translation) and is maximum, where images are rotationally aligned. The translation- and scale-invariancy are important features of a GI-based measure that can be used to find the rotational misalignment between the images in a robust and efficient manner. We will get back to this point when we describe GI-based registration methods in Section 3.

An example of the shape of MI functions computed using gradient intensities

and pixel intensities of two image is given in Fig. 3.4. We used images shown in Fig. 3.2(c) and Fig. 3.2(d) which are misaligned by 10° for these graphs. The MI function in the PI domain is smooth and correctly identifies the misalignment only because the misalignment is limited to the rotation parameter. If we introduce translation or scale misalignments the PI-based MI function will no longer identify the correct rotational misalignment since it is not translation and scale invariant. PI-based MI function has to be optimized simultaneously w.r.t. all the misalignment parameters. The MI function in the GI domain, also correctly identifies the rotational misalignment, but the shape of the function is not smooth. To find the optimal alignment, one has to perform an exhaustive search of the rotational parameter space. This is not a problem as the GI-based function is translation and scale invariant and only needs to be optimized w.r.t. to a single rotational parameter. The fact that we are able to find the misalignment by an exhaustive search is an advantage. It means that the GI-based method is inherently more robust as it is not subject to convergence to a local minimum which is a problem for local optimization methods such as Powell.

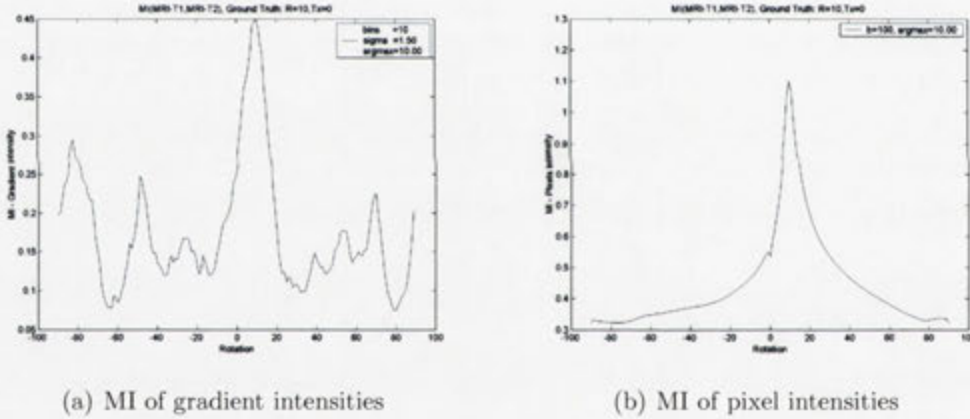


Figure 3.4: Sample MI functions for 2D images misaligned by 10° . The GI-based function is not as smooth, but this does not affect a registration algorithm based on the GI, as the global optimum is found by an exhaustive search of a 1D parameter space.

3.1.5 An Alternative Interpretation of 2D Gradient Intensity Based on Hough Transform

Vectors in a 2D gradient field can be parameterized by their angle θ and orthogonal distance from the origin ρ as shown in Fig. 3.5. Image gradients can be fully described by their spatial position (x, y) , magnitude and phase. We assume that

each gradient represents a line L parallel to the direction of the gradient vector which also passes through the gradient's spatial position as shown in Fig. 3.5. Each gradient vector is parameterized with its phase (angle of L) and the distance of L from the origin. This parametrization of the gradient field, drops the gradient magnitude and relaxes the spatial dependency of gradient vectors (all parallel gradient vectors on L are represented with the same parameters). We will see how this parametrization allows us to break down 2D rigid registration, which is normally a 3-parameter optimization problem, to three simpler 1-parameter optimizations.

Note that this method for gradient parametrization is slightly different from the standard Hough parametrization [38, 39] for lines and takes the direction into account. In gradient parametrization, ρ is a signed real number whereas in standard Hough transform ρ is non-negative. Using a signed ρ prevents formation of unwanted local minima in the cost function, that would otherwise be created due to ambiguity of the standard Hough parametrization of the lines that ignores their direction. We also use the gradient angle θ (e.g. θ_1 for \mathbf{g}_1 in Fig. 3.5) instead of the perpendicular line angle θ^\perp which is conventionally used in the standard Hough transform.

The Hough transformation of the gradient vectors, obtained in this manner, provides a spatially relaxed function of the image shape that can be used for registration of images. As we will see this transformation separates estimation of the rotation parameter from translation and scale parameters. Rotation, in the image domain, transforms to a circular shift in the Hough domain, which is invariant to the translation and scale between the images. This allows us to obtain a robust estimate of the rotation parameter quickly by performing an exhaustive search on a 1-parameter cost function that operates on a 1D data-set.

A similarity (rigid + isotropic scale) transformation of an image can be written as

$$\mathbf{x}' = s\mathbf{R}\mathbf{x} + \mathbf{t}, \text{ or} \quad (3.14)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = s \begin{bmatrix} \cos \gamma & -\sin \gamma \\ \sin \gamma & \cos \gamma \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \quad (3.15)$$

where t_x and t_y are translation parameters, γ is the rotation angle, and s is scale.

Let a gradient vector at spatial coordinates (x, y) and with angle θ (as shown in Fig. 3.5) be parameterized by (ρ, θ) , where

$$\rho = y \cos \theta - x \sin \theta, \quad (3.16)$$

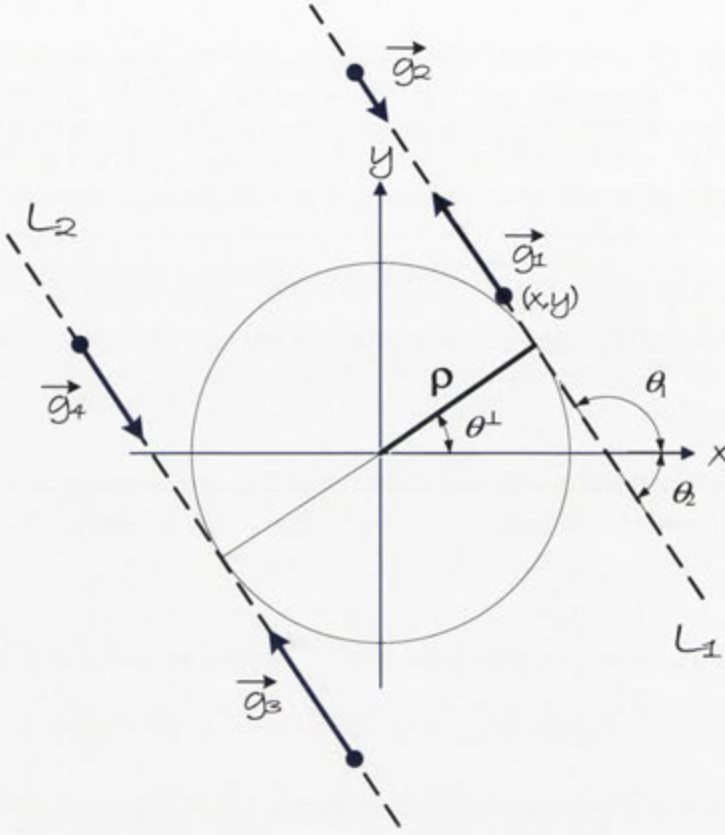


Figure 3.5: Gradient vectors in Cartesian coordinates and corresponding Hough space parameters. Each gradient vector is parameterized with its orthogonal distance ρ from the origin and its angle θ .

then the transformation results in the following change in Hough space

$$\rho' = y' \cos \theta' - x' \sin \theta', \quad (3.17)$$

$$\rho' = s[y \cos(\theta' - \gamma) - x \sin(\theta' - \gamma)] + \underbrace{t_y \cos \theta' - t_x \sin \theta'}_{\rho_t(\theta')}. \quad (3.18)$$

We now re-organize (3.18) so that it is comparable with the line equation in (3.16)

$$\frac{\rho' - \rho_t(\theta')}{s} = y \cos(\theta' - \gamma) - x \sin(\theta' - \gamma), \quad (3.19)$$

by comparing (3.19) with (3.16) we have

$$\begin{bmatrix} \rho' \\ \theta' \end{bmatrix} = \begin{bmatrix} s\rho + \rho_t(\theta + \gamma) \\ \theta + \gamma \end{bmatrix} \quad (3.20)$$

Equation (3.20) specifies a similarity transformation in Hough space. One immediate benefit is the separation of the rotation parameter from the translation and scale. Additionally, transformation in Hough space is computationally efficient as it consists of a shift along the θ axis and a multiplication and a constant shift for each value of θ along the ρ axis.

We now define the Hough transform function $\Gamma_H(\rho, \theta)$, as the number of gradient vectors at a particular direction θ and at a distance ρ from the origin. In practice, we need a discrete version of $\Gamma_H(\cdot)$. We use the *uniform volume histogram* (Section 3.1.6), which has been shown to be robust w.r.t. gradient calculation errors and noise and results in smoother cost function for image registration purposes. We will discuss how $\Gamma_H(\cdot)$ can be used for image registration in Section 3.4. We also note that the gradient intensity as defined in (3.7) can be written in terms of the Hough transform function $\Gamma_H(\cdot)$ as

$$\Gamma(\theta) = \int_{-\infty}^{\infty} \Gamma_H(\rho, \theta) d\rho. \quad (3.21)$$

3.1.6 Uniform Volume Histogram

For discrete images, (3.10) can be computed using a standard 2D histogram. The problem, however, is that a standard histogram representation is sensitive to the number of bins, gradient threshold selection, noise, quantization, and interpolation errors. This is further complicated by the fact that we are operating on a gradient field which has been numerically derived from a discrete image and is subject to a reduced SNR and errors in computation of its derivative components. It is, therefore, desirable to minimize additional side-effects of the histogram computation phase.

In [40], the authors propose a probability density estimation for 2D image pixel-intensities based on a continuous representation of image intensities and report improved robustness of the probability distribution function against noise and selection of the number of bins. The method in [40] assumes that the intensity in a triangle formed by three adjacent pixels can be estimated as a linear function of the intensity of the vertices. This in effect fits a plane to the vertices. This method could be adapted to improve gradient histograms as well, however, the extension to 3D volumes requires calculation of hyper-planes and simplex structures and is unnecessarily time consuming.

We introduce a histogram algorithm called uniform volume, which assumes a continuous representation for the gradient field $\mathbf{g}(\mathbf{x}) = [\rho_g(\mathbf{x}) \ \phi_g(\mathbf{x}) \ \theta_g(\mathbf{x})]^T$ exists locally within each cubic volume \mathbf{C} comprising 8 adjacent vertices of the

gradient image. Let $V = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_8\}$ represent the set of vertices. We assume that the gradient phase function within \mathbf{C} is bounded by the values of the function at the vertices of \mathbf{C} , i.e., for all $\mathbf{x} \in \mathbf{C}$

$$\begin{aligned} \min_{\mathbf{x} \in V} \phi_g(\mathbf{x}) &\leq \phi(\mathbf{x}) \leq \max_{\mathbf{x} \in V} \phi_g(\mathbf{x}), \\ \min_{\mathbf{x} \in V} \theta_g(\mathbf{x}) &\leq \theta(\mathbf{x}) \leq \max_{\mathbf{x} \in V} \theta_g(\mathbf{x}). \end{aligned} \quad (3.22)$$

Now consider a gradient intensity binning scheme with angular resolution of $2\delta\phi$ and $2\delta\theta$. Let (ϕ_i, θ_j) be the center of a bin which falls within the extents of the cubic volume \mathbf{C} , i.e.,

$$\begin{aligned} \min_{\mathbf{x} \in V} \phi_g(\mathbf{x}) &\leq \phi_i \leq \max_{\mathbf{x} \in V} \phi_g(\mathbf{x}), \\ \min_{\mathbf{x} \in V} \theta_g(\mathbf{x}) &\leq \theta_j \leq \max_{\mathbf{x} \in V} \theta_g(\mathbf{x}). \end{aligned} \quad (3.23)$$

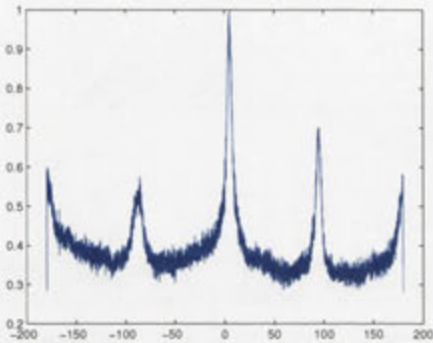
We assume the gradient field has the property that

$$\int \int \int_{\mathbf{C}} h_\phi(\mathbf{x}) h_\theta(\mathbf{x}) f_m(\mathbf{x}) dx dy dz \quad (3.24)$$

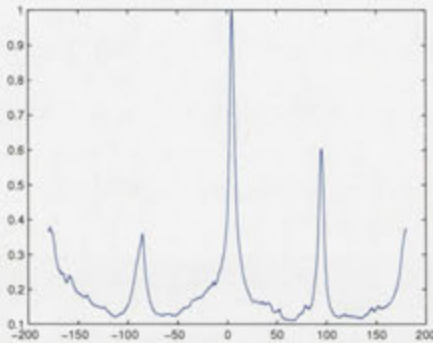
is the same for all 2D bin pairs represented by their center (ϕ_i, θ_j) , then the histogram contributions have to be uniformly distributed between bins that satisfy (3.23). Simply put, the condition in (3.24) allows us to uniformly distribute histogram contributions to the bins which fall within the extents of the cubic volume vertices.

One benefit of the uniform volume distribution as described above is that it is computationally inexpensive, since it does not explicitly require calculation of $\mathbf{g}(\mathbf{x})$ except at the bounding vertices. The histogram distribution favors areas of the image where the second derivative of the intensity is smaller or in other words the gradient function is smoother. This is in line with the findings in [40] for a robust histogram with less sensitivity to noise. We also designed the histogramming method to suppress the contributions of the areas of the image where there is a large discrepancy between the function values at adjacent vertices.

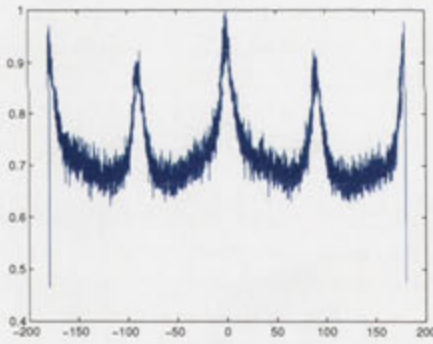
Compare the 1D histograms of gradient vectors' azimuth angle θ for an MR-T1 volume and its noisy version in Fig. 3.6, where white Gaussian noise with $\sigma = 0.1$ is added. Notice that uniform volume histograms are smoother and less affected by noise compared to standard histograms. These properties translate to smoother cost functions with fewer local minima, in the later stages of the algorithm (see Fig. 3.7), and allow our optimization method to converge more easily.



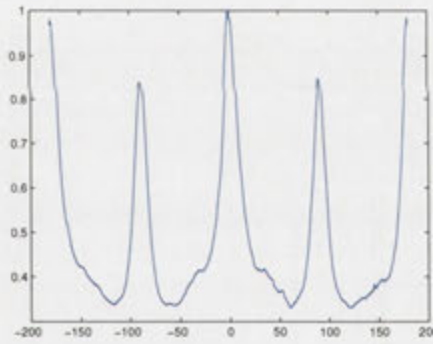
(a) Standard histogram



(b) Uniform volume histogram



(c) Standard histogram for a noisy image



(d) Uniform volume histogram for a noisy image

Figure 3.6: Uniform volume histograms exhibit smoother curves and better resistance to noise.

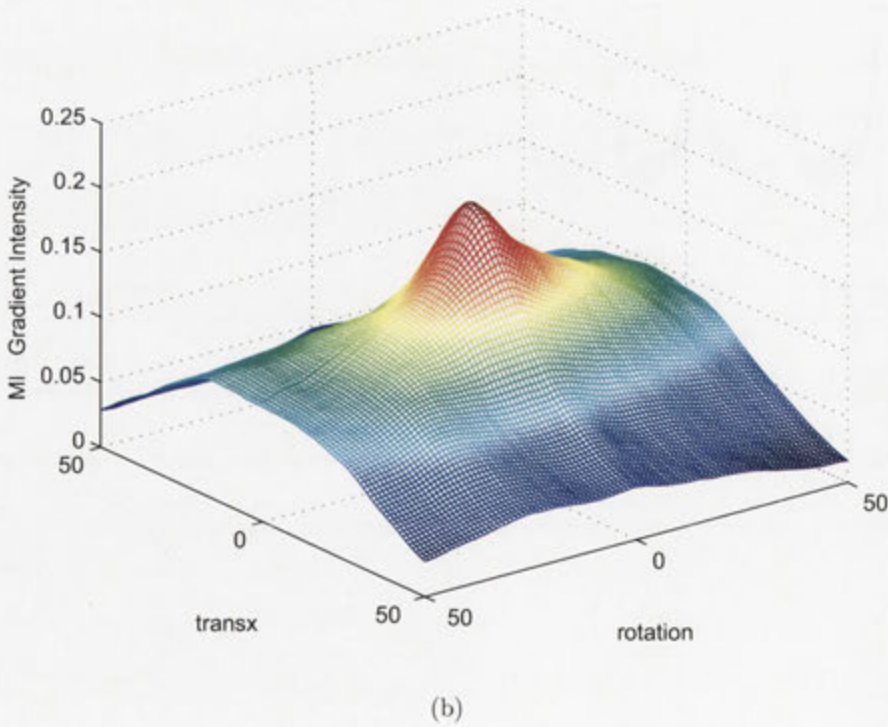
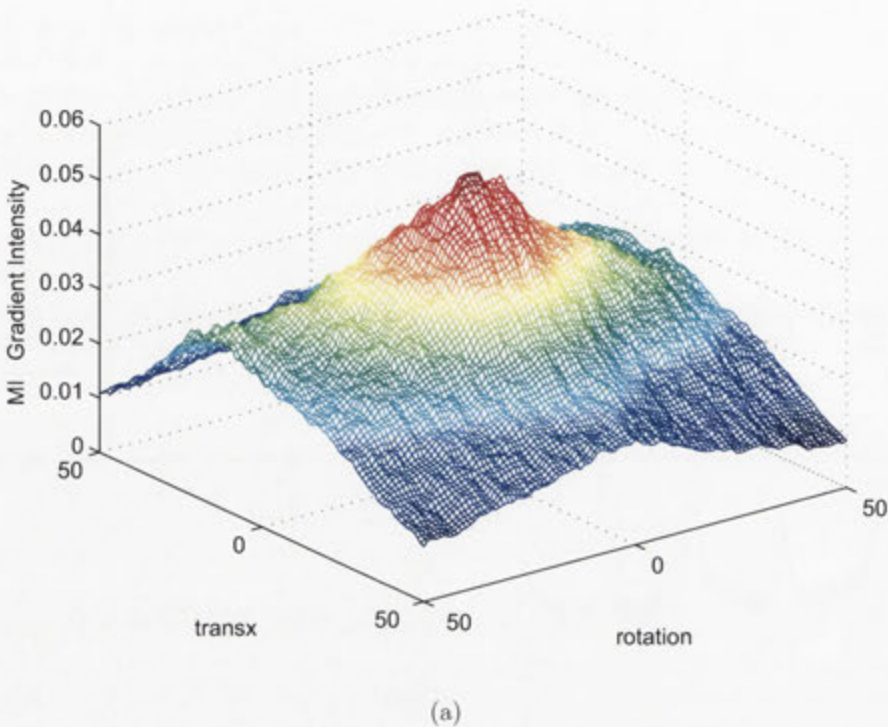


Figure 3.7: Comparison of 2D MI functions based on (a) standard and (b) uniform volume histogram. Uniform volume histogram results in a better-shaped and smoother cost function.

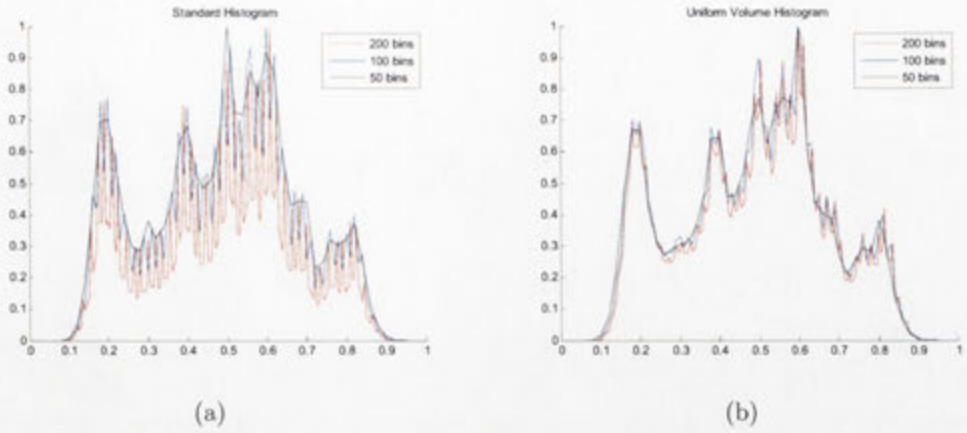


Figure 3.8: Comparison of 1D histograms with different number of bins based on (a) standard and (b) uniform volume histogram. Uniform volume histogram maintains a relatively consistent shape across a wide range of bins.

We implemented the uniform volume histogram method in a circular fashion to accommodate rollover of angles. The angular resolution used for both zenith and azimuth angles is 1.0° .

Another benefit of UVH is its robustness w.r.t. the number of bins. Compare the 1D and 2D histograms computed with the standard histogram and the UVH in Fig. 3.8 and Fig. 3.9. The distribution of a standard histogram varies greatly with the number of bins, whereas the UVH maintains similar distribution for a wide range of bins. This is particularly evident from the 2D histograms which are shown separately in Fig. 3.9.

3.2 2D Registration

3.2.1 Estimating the Rotation Parameter

The gradient phase matrix is converted into a histogram at a desired angular resolution using (3.7). The histogram is then normalized by the most frequent sample to create the gradient intensity vector as in (3.9). Regardless of the size of an image or its dimensionality, we always calculate MI for two $1 \times d$ vectors: a fixed vector and a moving vector, where d is the number of directions. We use the angular resolution of 1° , which limits the gradient intensity to a vector of size 1×360 .

The MI function is computed by iterating over the rotational range. At each step, a circular shift is applied to the moving GI vector and the MI between the resulting vector and the fixed vector is calculated using (2.17). This has a significant computational advantage over the PI method, where at each step one would require

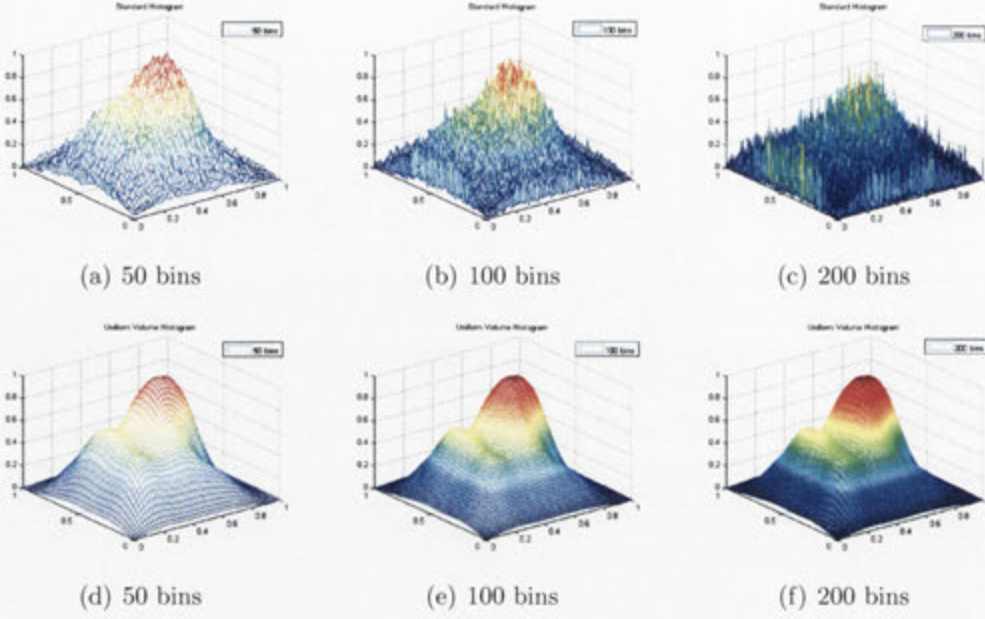


Figure 3.9: Comparison of 2D histograms with different number of bins based on (a-c) standard and (d-f) uniform volume histogram. Uniform volume histogram maintains a relatively consistent shape across a wide range of bins.

to rotate the moving image itself and calculate MI for the entire image. MI for the gradient intensity of the images can be computed under $1ms$ on a standard PC (using a single-core). This allows us to perform an exhaustive search of the rotation parameter's dynamic range to find the optimal rotation where the cost function attains its global minimum.

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} -\mathcal{S}_{\text{MI}}(V_{\mathcal{F}}; V_{\mathcal{M}(\theta)}) \quad (3.25)$$

where $V_{\mathcal{M}(\theta)}(\phi) = V_{\mathcal{M}}(\phi - \theta)$, and $V_{\mathcal{F}}$ and $V_{\mathcal{M}}$ are the GI vectors for fixed and moving images, respectively.

GI vectors have only 360 elements. As such, the number of samples available for a GI histogram is indeed very small. This means that to get a meaningful histogram we have to use a small number of bins. In our experiments, we found using 10 bins for pmf estimations is a reasonable choice.

3.2.2 Estimation of Scale and Translation

2D rigid transformation of a vector \mathbf{x} can be written as

$$\mathbf{x}' = s\mathbf{R}\mathbf{x} + \mathbf{t}, \quad (3.26)$$

where \mathbf{x}' is the transformed vector, s is the scale factor, \mathbf{t} is the translation vector, and \mathbf{R} is the rotation matrix. The distance between two points \mathbf{x}_1 and \mathbf{x}_2 is defined as $\mathbf{d}(\mathbf{x}_1, \mathbf{x}_2) = \|\mathbf{x}_1 - \mathbf{x}_2\|_2$, where $\|\cdot\|_2$ is the Euclidean ℓ_2 norm. The transformation is distance preserving up to a scale parameter

$$\mathbf{d}(\mathbf{x}'_1, \mathbf{x}'_2) = s \|\mathbf{R}(\mathbf{x}_1 - \mathbf{x}_2)\|_2 = s\mathbf{d}(\mathbf{x}_1, \mathbf{x}_2). \quad (3.27)$$

An estimate of the scale parameters can be obtained using the centroid of image gradient maps and their distance from other gradients in the each image

$$\hat{s} = \frac{\sum_{\mathbf{x}' \in G'} \mathbf{d}(\mathbf{x}', \mathbf{x}'_c)}{\sum_{\mathbf{x} \in G} \mathbf{d}(\mathbf{x}, \mathbf{x}_c)}, \quad (3.28)$$

where \hat{s} is the estimated scale parameter and G and G' are the gradient maps of fixed and moving images, respectively.

Given the rotation and scale parameters, the translation can be found from (3.26) by replacing \mathbf{x} and \mathbf{x}' with the centroid of the fixed and moving images, respectively

$$\hat{\mathbf{t}} = \mathbf{x}'_c - s\mathbf{R}\mathbf{x}_c. \quad (3.29)$$

The centroid-based estimation of scale and translation is noisy and deteriorates for images with less overlapping regions. We give a robust method for estimation of scale and translation using the Hough transform in Section 3.4. However, in the experiments of this section we demonstrate that a good estimation of the rotation parameter, even without a robust estimation of scale and translation, improves the capture range and the overall robustness of image registrations.

To obtain accurate registration results a final local optimization round using the pixel intensities needs to be performed. We use Powell's multi-dimensional direction set algorithm to find the optimal alignment. The optimization method is initialized from the estimates of transformation parameters previously obtained using gradient intensity of the images.

One problem with the Powell algorithm is that from an observer's point of view who knows where the minimum is located, it appears to spend a lot of time, iterating on and around the minimum. Obviously, the only way the optimization algorithm can satisfy itself that it has found the actual minimum is to spend enough time to check the surroundings. Since we already know that we are able to initialize the algorithm close to the optimal alignment we propose a modification to the algorithm to improve convergence rate as described in the following section.

3.2.3 Enhanced Powell Optimization

Powell's multi-dimensional direction set algorithm finds the minimum of a cost function by iteratively minimizing the function along a set of N directions, where N is the number of independent parameters of the cost function. A line minimization algorithm (typically Brent's one dimensional line minimization algorithm [30]) is used to find the minimum in a given direction.

In a standard Powell implementation, such as given in [22], fractional and absolute tolerance parameters are used to control convergence of line minimizations. Reducing the tolerance parameters, generally, results in better accuracy at the expense of a slower convergence rate (increased computational cost). In our experience with Powell for image registration, we observed that when the method converges to the correct alignment, it spends a considerable amount of time checking the perimeter of the optimal solution before terminating the search. This naturally suggests that the tolerance can be increased to improve the convergence rate. However, we also observed that increasing the tolerance may reduce the capture range of the method and increase the chance of convergence to local minima resulting in incorrect alignment of images. Lack of a principled method for choosing the tolerance parameters makes it difficult to strike a right balance between the convergence rate and accuracy/correctness of the registrations.

We introduce additional flexibility in controlling Powell's convergence through the use of a set of resolution parameters for each parameter of the cost function being optimized. Our modified Powell optimization maintains a list of previously evaluated points and will only evaluate the cost function at a new point in space if the absolute distance of the two points for each dimension exceeds the specified resolution in that dimension. This allows us to specify small tolerance values without necessarily increasing the computational cost. Let δ be the resolution vector whose elements correspond to the desired resolution for each dimension of the optimization parameter space, and \mathcal{P} the set of previously visited points. The cost function at a new point \mathbf{q} is evaluated if

$$|q_i - p_i| > \delta_i, \quad 1 \leq i \leq N, \quad \forall \mathbf{p} \in \mathcal{P}. \quad (3.30)$$

The method returns the cost associated with the closest evaluated point otherwise. We note that our method is equivalent to the standard Powell optimization for $\delta = \mathbf{0}$ and can be implemented with minimal modification to a standard implementation.

Resolution parameters allow us to independently control the accuracy for each

parameter being optimized. Use of an appropriate resolution allows the method to converge much more quickly than a standard Powell implementation without affecting the accuracy of registrations as demonstrated by the experiments in Section 3.3. The resolution parameters can be set based on the desired precision for a given parameter. This allows for an intuitive selection of the resolution parameters. As a rule of thumb a resolution parameter within 1-2 orders of magnitude smaller than the required precision results in a reasonable balance between the accuracy and the convergence rate.

3.3 Experiments

We tested our method on 2D aerial images and 3D simulated MR images of brain generated by Brain Web [41]. The experiments on MR images are to demonstrate the applicability of the method for multi-modal images and to motivate further extension of the method to support full 3D registration which is the subject of Section 3.5. We calculated the registration error for 2D images by selecting 10,000 control points uniformly distributed across the image and calculating the average distance between the transformed control points under the calculated transformation and the actual transformation.

3.3.1 Registration of 2D Images

We compared GI method with the conventional PI method by conducting two sets of experiments on aerial images. For each run we randomly created 500 transformation matrices within the dynamic range of the parameter space. The reference image was transformed by each matrix and GI and PI registration was performed on the reference and the transformed images. A smaller dynamic range was used for the first set of experiments and the dynamic range was increased for the second round as shown in Table 3.3.1. The GI method was run with three Powell resolutions high, medium and low (0.001, 0.01 and 0.1 pixels). The results are shown in Fig. 3.10 (darker colors are associated with the first experiment and lighter colors with the second experiment).

The robustness and performance of registration significantly improved under the GI method as shown in Fig. 3.10. We declared a registration failed if the average pixel alignment error was more than two pixels. As shown in Fig. 3.10(a), the PI method performance is reasonable in the first experiment but as we increase the dynamic range in the second experiment the performance drops below 20% since the chance of being trapped by local minima increases. The GI method, on the

Table 3.3: Dynamic range of parameters used in the experiments

	Round One Tests	Round Two Tests
Translations	$[-25 \quad 25]$	$[-50 \quad 50]$
Rotation	$[-25^\circ \quad 25^\circ]$	$[-180^\circ \quad 180^\circ]$
Scale	$[0.75 \quad 1.25]$	$[0.5 \quad 1.30]$

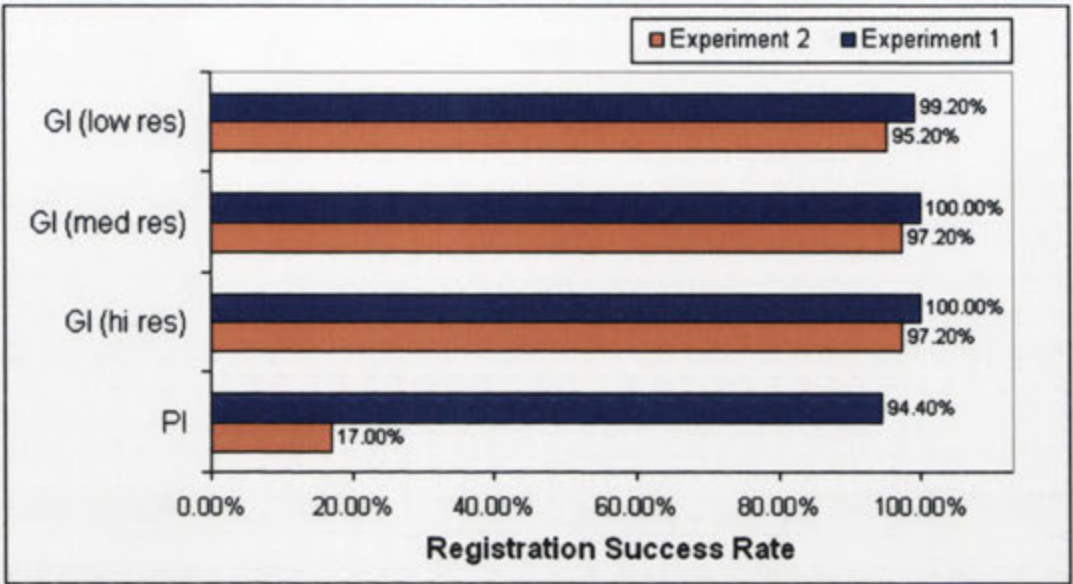
other hand, performs very well in both cases with performance levels more than 99% and 95% for experiment one and two, respectively. Also notable is the success rate of GI method at medium resolution which successfully registered all images in the first experiment and 486 out 500 in the second experiment. This is due to the good initial estimates of transformation parameters under the GI method. Throughout our experiments estimation of rotation and scale parameters were very close to the actual alignment with a mean error of 0.54° and 0.1% for rotation and scale, respectively. However, the translation estimations which are mainly based on centroids are less accurate with an average error of 4.1 pixels and deteriorate under larger misalignment between the images (centroid calculation is sensitive to partial overlap). This explains the slight performance drop in GI method in the second experiment.

In addition to improved performance, GI-method improves efficiency by allowing the optimization step to converge more quickly as shown in Fig. 3.10(b). Obviously the efficiency can be improved by reducing the resolution of the algorithm. Based on experiments a medium choice of resolution seems to provide a good balance between the accuracy and efficiency.

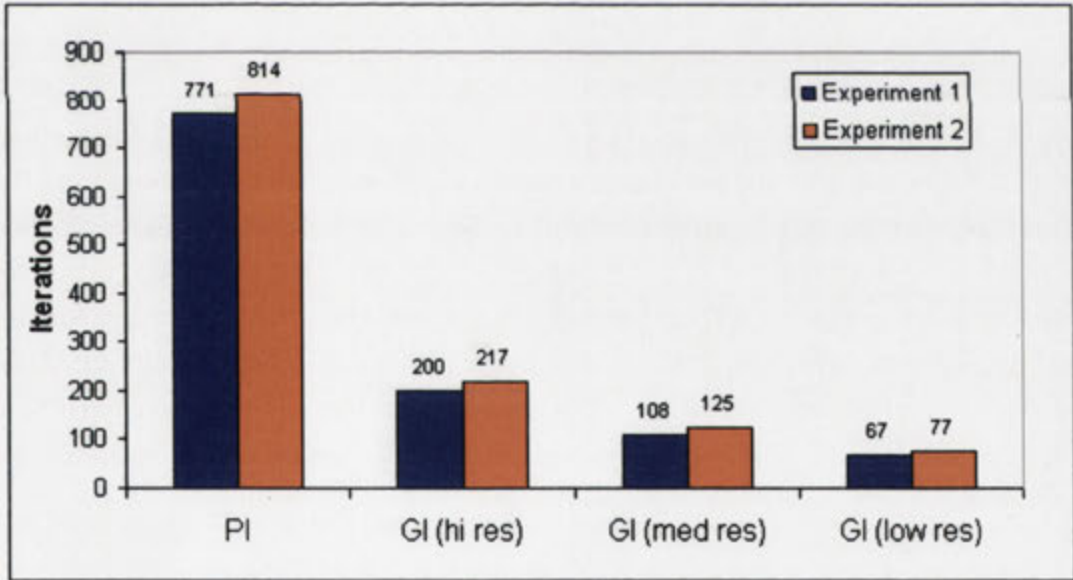
3.3.2 Registration of Multi-Modal Images

We experimented with several combinations of multi-modal MR images of brain to determine the applicability of our method in the presence of non-linear intensity variations. Fig. 3.12 shows a sample of T1, T2 and PD modalities used in our experiments [41]. The transformation were applied to the entire volume, however the misalignment between the images were limited to in-plane transformations. We will look at full 3D-3D registrations in Section 3.5.

Fig. 3.11 demonstrates that our optimization method (referred in the image as ‘guided optimization’) improves performance and efficiency of the registration task compared to an optimization algorithm initialized from the origin of the parameter space (referred in the image as ‘blind optimization’). The results in Fig. 3.11 are



(a) Registration success rate for each method. Conventional PI method performance drops below 20% for larger dynamic ranges, while GI method sustains more than 95% success rate.



(b) Iterations required for convergence of the optimization algorithm for each method. The GI methods converge much more quickly.

Figure 3.10: Improved performance and efficiency of the GI method.

given for registration of T1 to T2 images of the brain. In this experiment, T2 was translated by $(-20, 20)$ mm and scaled by 0.9, the rotation parameter was varied from -50° to 50° in 10° increments. The efficiency of the registration improved on average by a factor of 12. The GI method was able to achieve accuracy well below the voxel size of 1 mm with an average error of 0.18 mm.

3.3.3 Discussion

Accuracy of Estimating Rotation

Average error in estimating the rotation parameter for 2D experiments was 0.54° and for 3D experiments was below 1° (refer to Fig. 3.13), which is within the angular resolution of our experiments. This demonstrates that mutual information of gradient intensity is a good measure of directional similarity, irrespective of the translation and scaling between the images.

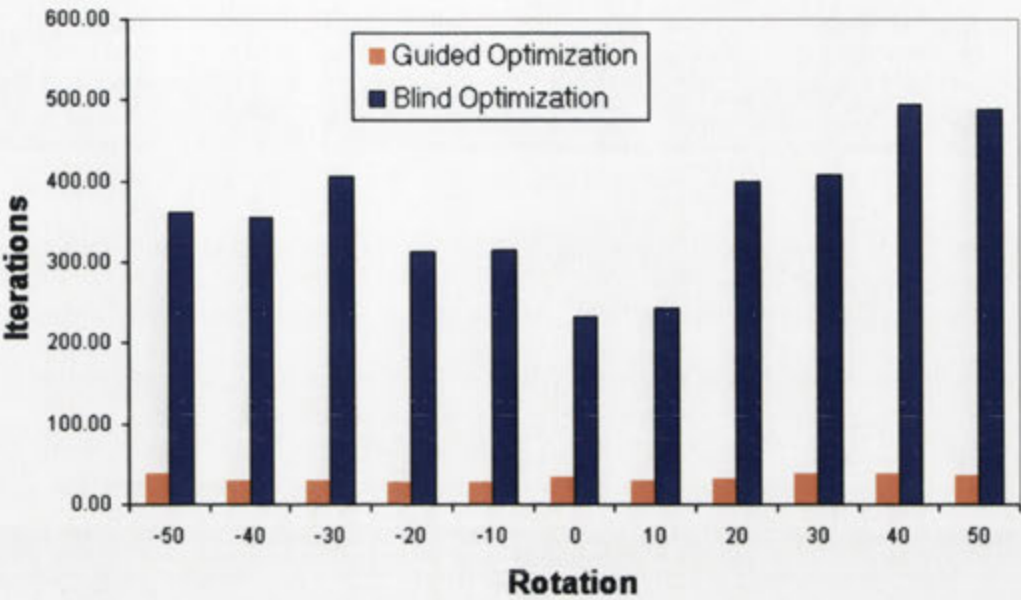
Invariance to Translation and Scaling

When fixed and moving images are scaled by a factor within a reasonable range or translated, their shape remains unchanged. As such, one would expect that the directional content of image gradients be relatively maintained.

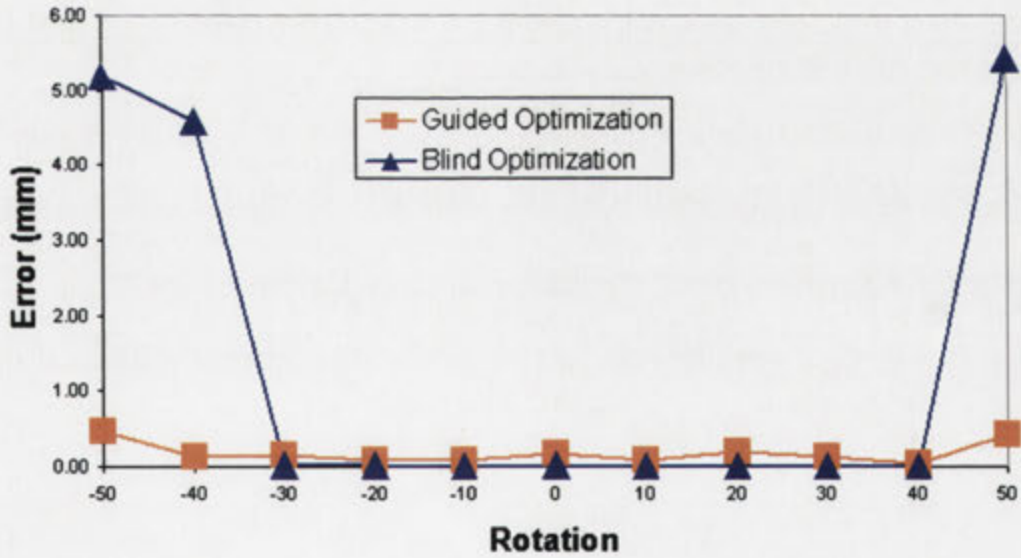
Fig. 3.13 underlines the invariance of the GI method to scale and translation. Fig. 3.13(a) shows a number of experiments, where an MRI-T2 is scaled from 0.8 to 1.25 while being rotated between 0° to 30° . The transformed image is then registered to a reference MRI-T1. The experiments show that GI method estimates the rotation parameter with an average error of 0.6° . Fig. 3.13(b) shows a number of experiments, where an MRI-T2 is translated from $(-40, 40)$ to $(40, 40)$ while being rotated between 0° to 30° . The transformed image is then registered to a reference MRI-T1. The experiments show that GI-based method estimates the rotation parameter with an average error of 0.9° .

Robustness with Respect to Partial Overlap

In some of our experiments clipping of the transformed images removed a significant number of important gradient vectors on the outer boundary of the image (e.g. Fig. 3.12(c)), however a close estimate of the rotation parameter could still be found, which demonstrates the robustness of the gradient intensity method for partially overlapping images.



(a) More than an order of magnitude improvement in efficiency (execution times are proportional to iterations).



(b) The registration error for our method remains nearly constant and is well below the voxel size; the conventional method fails for rotations outside $\pm 40^\circ$.

Figure 3.11: Superior robustness and improved efficiency of our method compared to the conventional method.

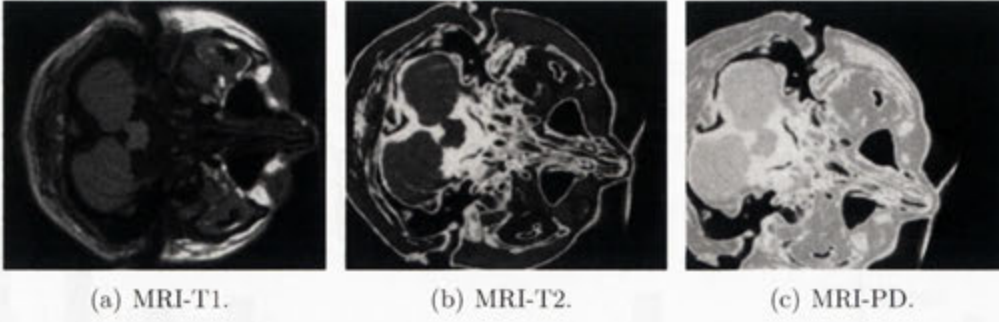


Figure 3.12: A sample set of synthetic MR images of brain used in our experiments with voxel size of 1 mm^3 .

Guided Optimization vs Blind Optimization

The guided optimization outperforms blind optimization both in terms of efficiency and accuracy. The registration error for the guided optimization remains almost constant (e.g. Fig. 3.11(b)). While the average registration error is lower for the guided optimization, the minimum registration error can be higher for guided optimization at lower misalignments, unless a sufficiently high resolution is used. The choice of resolution for the guided optimization is determined by the minimum accuracy required by the application. For example for most medical application sub-voxel accuracy is required which in our experiments could be achieved by a medium choice of the resolution parameter.

3.4 2D Registration in Hough Space

3.4.1 Estimation of Transformation Parameters

Let $\Gamma'(\cdot)$ be the gradient function of the transformed image in the Hough domain as defined in Section 3.1.5. Using (3.20) we have

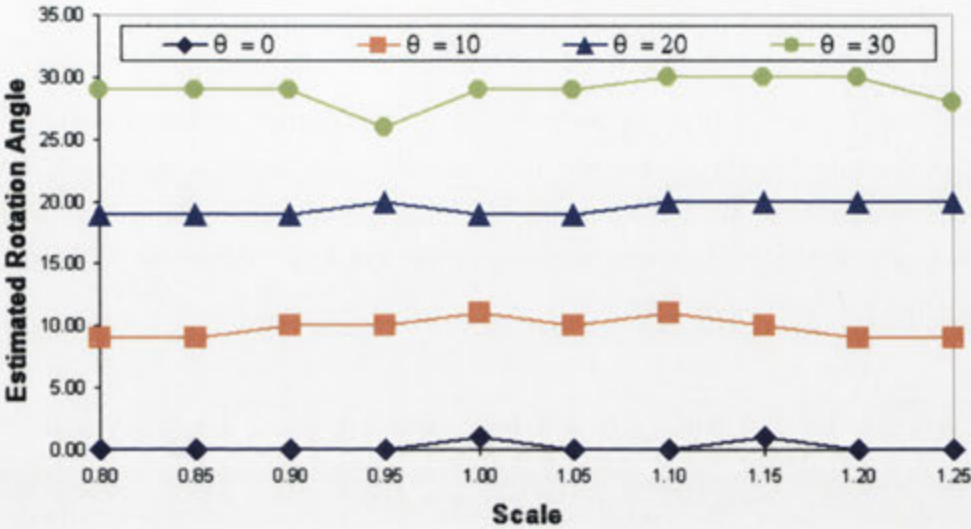
$$\Gamma'(\rho, \phi) = \Gamma(s\rho + \rho_t, \phi + \gamma), \quad (3.31)$$

$$\sum_{\rho} \Gamma'(\rho, \phi) = \sum_{\rho} \Gamma(s\rho + \rho_t, \phi + \gamma), \quad (3.32)$$

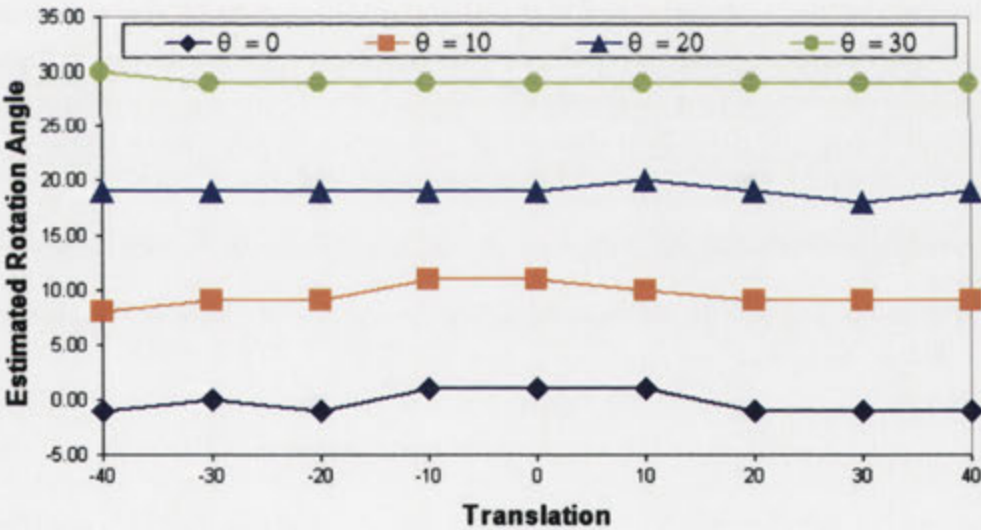
$$\text{or } \bar{\Gamma}'(\phi) = \bar{\Gamma}(\phi + \gamma), \quad (3.33)$$

where $\bar{\Gamma}(\cdot)$ and $\bar{\Gamma}'(\cdot)$ are the average of gradient functions along the ρ axis.

Equations (3.31) and (3.32) allow us to decouple the estimation of the transformation parameters. It can be readily seen that the rotation parameter reduces to a shift along the ϕ axis in the Hough domain and is independent of the scale



(a) Estimated rotations for various misalignments vs. scale



(b) Estimated rotations for various misalignments vs. translation

Figure 3.13: Estimation of rotation parameter using GI method demonstrating invariance to scale and translation. The legend shows ground truth for each experiment.

and translation parameters. As such, finding the rotation parameter between the images reduces to a 1-parameter optimization problem over the rotational dynamic range of $(-\pi, \pi]$

$$\hat{\gamma} = \underset{\gamma}{\operatorname{argmin}} -\mathcal{S}_{\text{MI}}(\overline{\Gamma}'(\phi); \overline{\Gamma}(\phi + \gamma)). \quad (3.34)$$

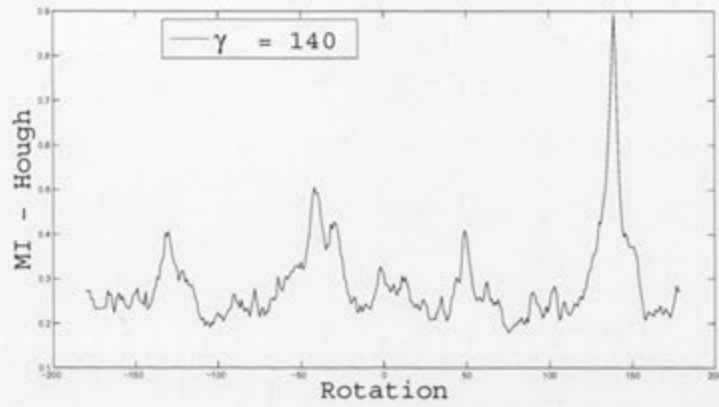
In practice, due to the small overhead in calculating the 1D cost function, we use an exhaustive optimization strategy by calculating the cost function at 1° intervals and choosing the rotation parameter which results in the minimum cost. Resolving γ is an important step, since multi-resolution methods perform much better if the images are rotationally aligned.

We can further exploit Hough space transformation to quickly and robustly estimate translation parameters. For rigid registration ($s = 1$), we pick a number of lines parallel to the ρ axis and calculate MI for all values of ρ_t in the dynamic range at 1 pixel intervals. MI calculation is very efficient since we are computing the similarity for two 1D data-sets (two lines in Hough space). Let us pick $\phi = \pi/2$ line for which $\rho_t = t_x$. As shown in Fig. 3.14(b), the MI function attains its maximum at t_x . Similarly, we could estimate t_y by looking at $\phi = 0$ line. However, this may not always be robust as demonstrated in Fig. 3.15. We can improve the robustness of the estimation by looking at the similarity functions of other lines, which provide combined estimates of t_x and t_y ($\rho_t(\phi) = t_y \cos \phi - t_x \sin \phi$).

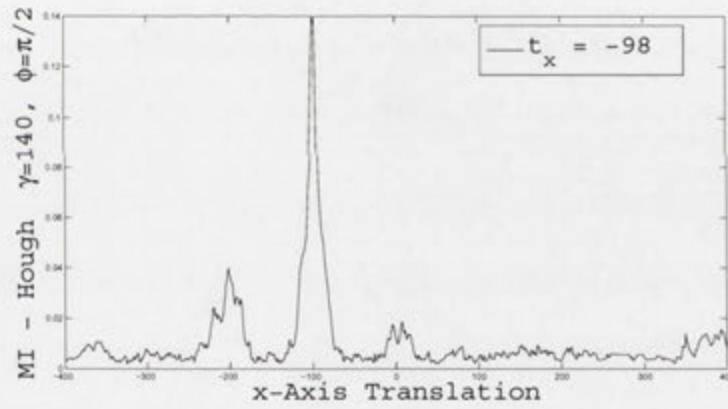
Let us treat the MI function as the probability distribution of the random variable ρ_t . For each line $\phi = \alpha$ we have

$$\Pr(\rho_t(\alpha)) \propto \mathcal{S}_{\text{MI}}(\Gamma'(\rho, \alpha); \Gamma(\rho + \rho_t(\alpha), \alpha + \hat{\gamma})). \quad (3.35)$$

Equation (3.35) may be thought of as a projection of the probability distribution $\Pr(\rho_t)$ in the direction determined by $\phi = \alpha$. Our aim is to find the best (most probable) estimate of the translation parameters t_x and t_y given a number of projections of $\Pr(\rho_t)$ in different directions. This can be achieved by a process of back-projection analogous to that used in computing the inverse Radon transform. We back-project $\Pr(\rho_t)$ for a number of lines, and find the estimate of translation parameters where the back-projected probability density function achieves its maximum, as shown in Fig. 3.15. In practice, no more than 10 lines were required for achieving robust results.

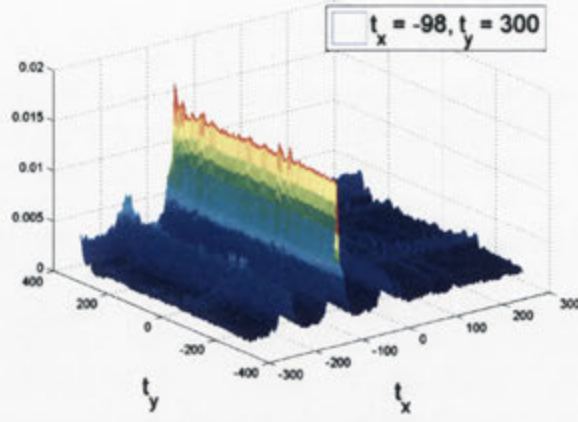


(a) 1D MI function (rotation)

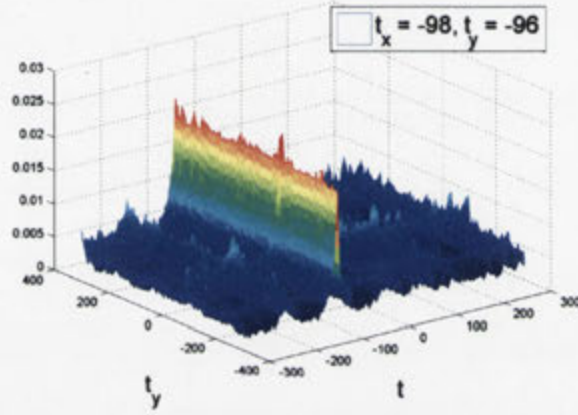


(b) 1D MI function (x-axis translation)

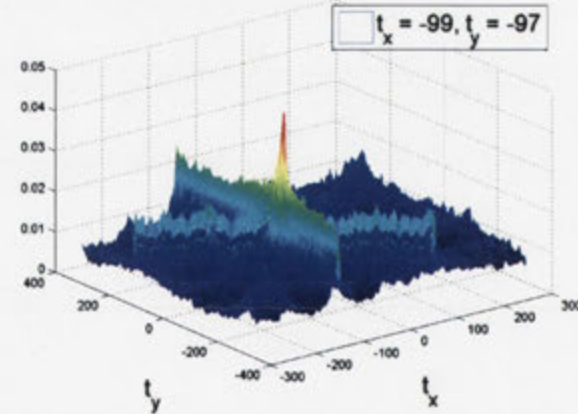
Figure 3.14: 1D MI functions are shown for images of Fig. 3.16(a). The misalignment between the images is $[-98.03 \ -97.00 \ 140.52^\circ]$. Our method estimates the transformation parameters as $[-98 \ -97 \ 140^\circ]$. Note that the shape of MI functions are unimportant at this stage, since we are performing an exhaustive search to estimate the parameters.



(a) Estimation with 2 lines



(b) Estimation with 4 lines



(c) Estimation with 8 lines

Figure 3.15: Robustness of estimating translation parameters improves by adding more lines. (a) With 2 lines, t_y is estimated incorrectly. (b) With 4 lines both parameters are correctly estimated but there are other high peaks. (c) With 8 lines, a single dominant peak is formed at the correct alignment.

3.4.2 Final Optimization

As the final step, we switch back to the intensity domain to obtain more accurate results. We use a multi-resolution Gaussian pyramid which is optimized using the Simplex [42] method. The multi-resolution optimization is initialized with the estimated parameters. The λ parameter which determines the volume of the initial simplex is set to a small value (10 pixels along the translation axis and 10° along the rotation axis) to allow the optimization to converge quickly.

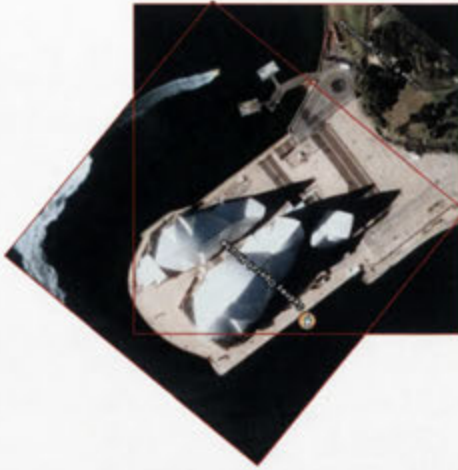
3.4.3 Experiments

We tested our method on aerial images and 2D multi-modal medical images. For aerial images, we used 20 scenes in our experiments selected from four categories: landmark, urban, rural and natural. *Landmark* images include a prominent man-made structure, *urban* images are taken from city centers and mostly contain rectilinear structures, *rural* images contain few buildings and are otherwise occupied by natural scenery, and *natural* images contain no artificial structures or roads. One sample registration per category is shown in Fig. 3.16. For medical images, we experimented with CT and MR slices of the brain. The medical images were already registered w.r.t. out-of-plane parameters and the method was used to recover in-plane misalignments in the transversal plane. A sample pair of images with their initial alignment and the checkerboard overlay of the images prior and after registration is shown in Fig. 3.17. The checkerboard image after registration clearly shows the skull boundary aligned.

In total, we performed 8000 registrations in two sets of experiments by generating random transformations for each image within the angular dynamic range of $(-\pi, \pi]$. The translational dynamic range was set to $[-100, 100]$ pixels for the first experiment and increased to $[-150, 150]$ for the second experiment. The minimum overlap between the images was 50% and the average overlap was 75% for the entire set.

We compared our method with the conventional multi-resolution method in the pixel intensity domain. We used MI as the similarity measure and optimized the cost function using the Simplex method.

The registration errors were calculated by selecting 10,000 control points uniformly distributed across the image and computing the average distance between the transformed control points under the calculated transformation and the actual transformation.



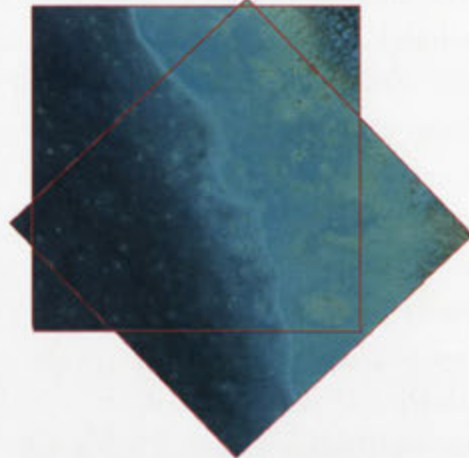
(a) Landmark - Sydney Opera House



(b) Urban - Los Angeles



(c) Rural - Farm



(d) Natural - Coral Sea



(e) House by the Lake



(f) Café

Figure 3.16: (a-d) Examples of registered images in each category. (e-f) Pictures taken with a hand-held camera. There are some affine and perspective distortions between the images, however our method can find a rigid approximation.

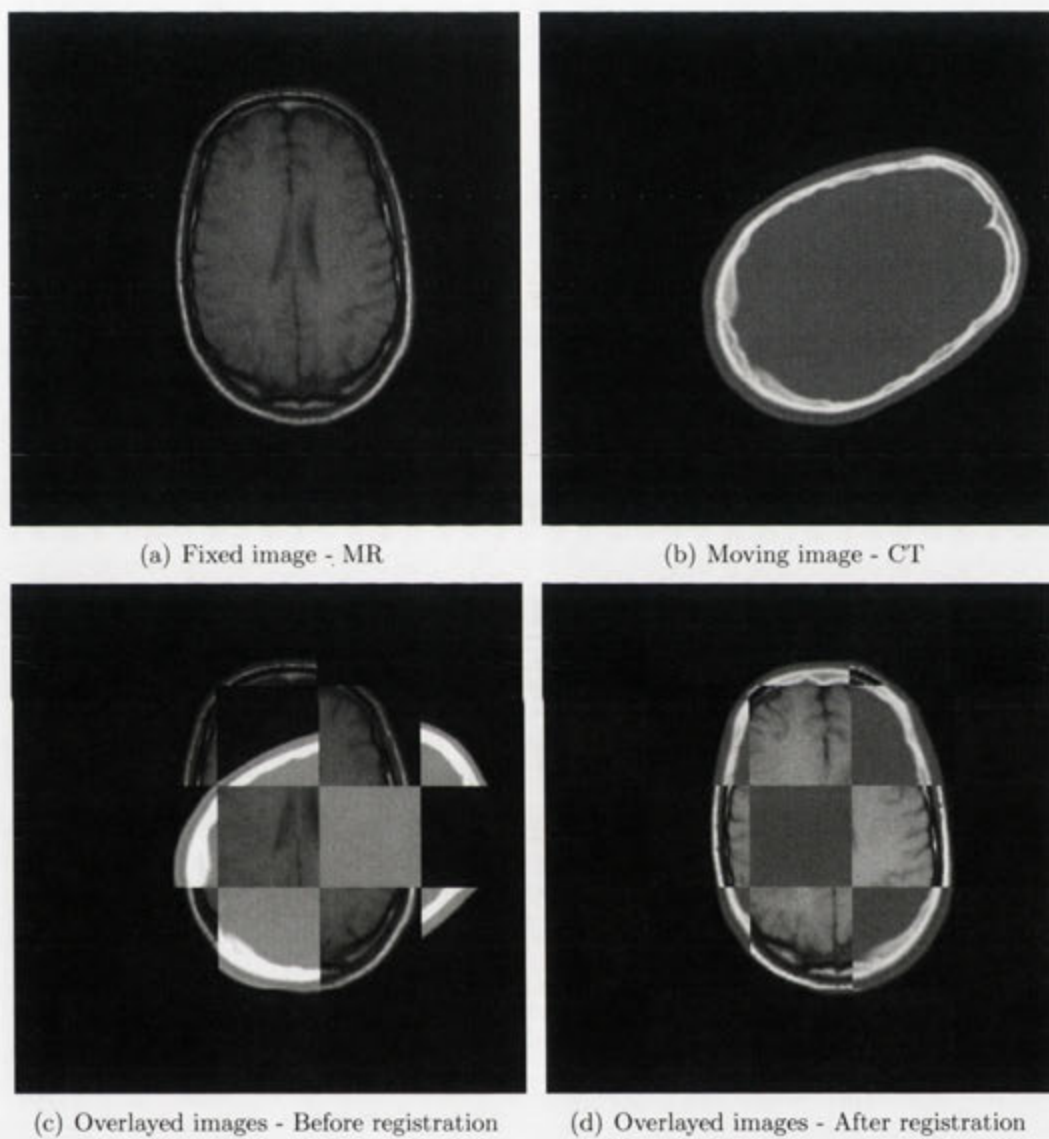
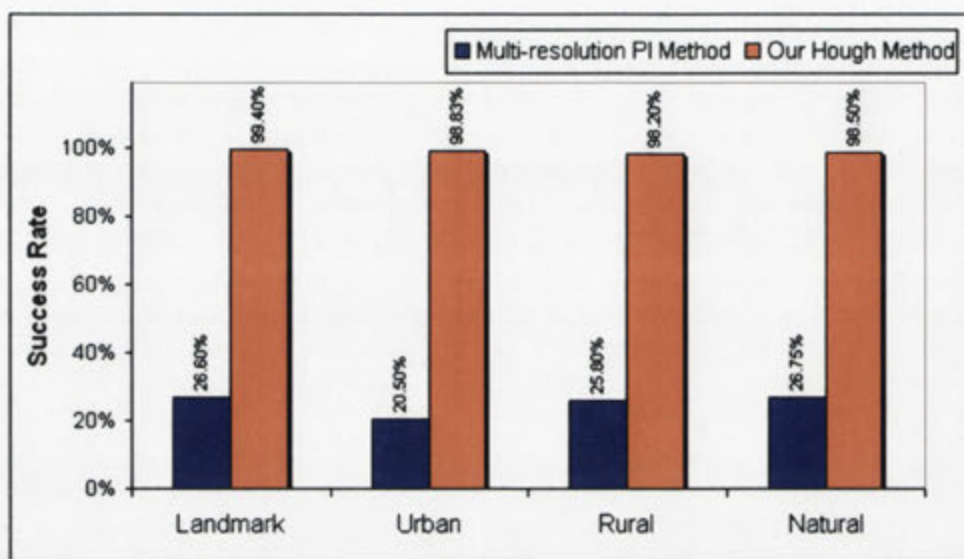
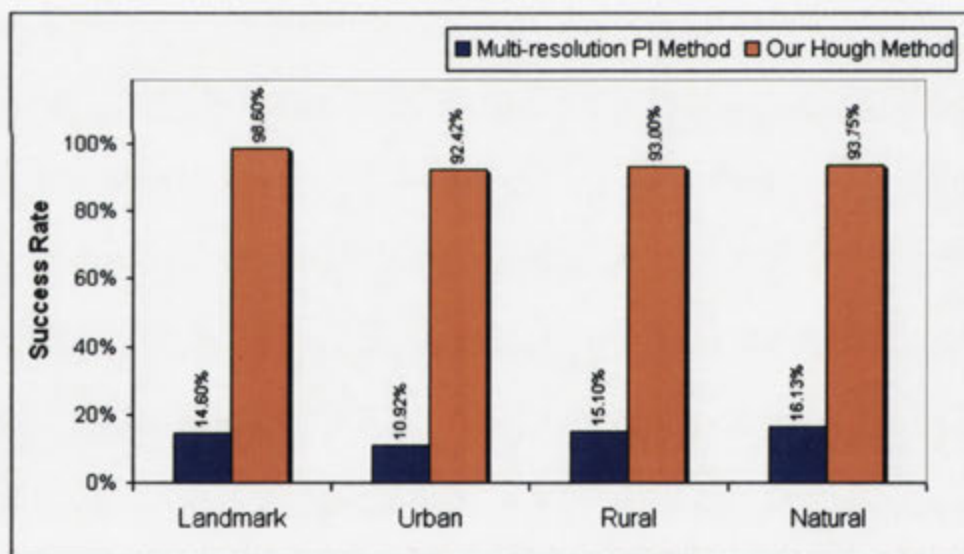


Figure 3.17: Sample 2D multi-modal images registered using Hough transform of the gradients.



(a) Experiment 1



(b) Experiment 2

Figure 3.18: Superior performance of our registration method compared to the conventional multi-resolution approach.

3.4.4 Discussion

Accuracy and Performance of the Registrations

The success rate of our method was 99% for the first experiment and 95% for the second experiment. Whereas the performance of the conventional multi-resolution method was 24% and 15%, respectively. The performance of our method decreases as we increase the translation dynamic range, this is due to the reduction in overlap between the images and deterioration of the rotation estimate. The poor performance of the conventional method is due to its inability to cope with the large capture range of the experiments. Our method performed very well for overlaps as low as 50%, for all groups of images. We also tested our method with images taken with a hand-held camera and without a tripod. The images taken in this way contain perspective deformations, however our method is able to find a close rigid approximation as shown in Fig. 3.16(e) and Fig. 3.16(f).

Both Hough-based and intensity-based methods registered images very accurately, whenever they were initialized within the capture range of their respective cost functions. The average registration error was 0.42 pixels for our method. This was expected, since MI-based cost functions are known to be accurate [43].

Capture Range of the Hough Method

Cost functions based on Hough gradient functions exhibit larger translational capture range compared to intensity-based cost functions. The capture range is typically more than twice, as shown schematically in Fig. 3.19.

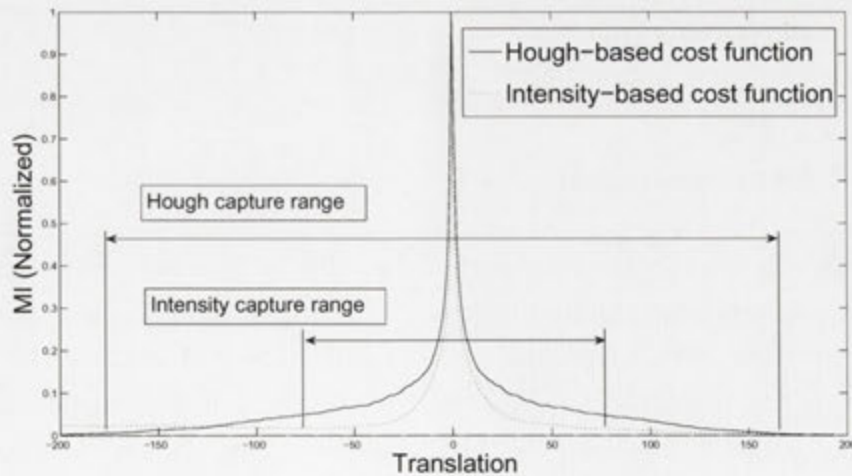


Figure 3.19: Hough method has a larger translational capture range, due to the spatially relaxed formulation of the cost function.

The larger capture range can be attributed to the spatially relaxed formulation of the gradient function Γ . The gradients can be freely displaced along the line that passes through them without affecting the gradient function. For example, in Fig. 3.5, placing g_1 anywhere on L_1 does not change Γ .

Similarity Registration

It is possible to use our Hough registration method for similarity registration. The translation and scale can be estimated using a method similar to that described in Section 3.4.1. However this time an exhaustive 2D search needs to be performed.

$$[\hat{t}_y, \hat{s}] = \underset{t_y, s}{\operatorname{argmin}} -\mathcal{S}_{\text{MI}}(\Gamma'(\rho, 0); \Gamma(s\rho + t_y, \hat{\gamma})), \quad (3.36)$$

$$[\hat{t}_x, \hat{s}] = \underset{t_x, s}{\operatorname{argmin}} -\mathcal{S}_{\text{MI}}(\Gamma'(\rho, \frac{\pi}{2}); \Gamma(t\rho - t_x, \frac{\pi}{2} + \hat{\gamma})). \quad (3.37)$$

Alternatively, one can optimize for two translations and the scale parameter, once the rotation is resolved, using a local optimization algorithm such as Simplex:

$$[\hat{t}_x, \hat{t}_y, \hat{s}] = \underset{t_x, t_y, s}{\operatorname{argmin}} -\mathcal{S}_{\text{MI}}(\Gamma'(\rho, \phi); \Gamma(\rho', \phi')), \quad (3.38)$$

$$\rho' = s\rho + t_y \cos(\phi + \hat{\gamma}) - t_x \sin(\phi + \hat{\gamma}), \quad (3.39)$$

$$\phi' = \phi + \hat{\gamma}. \quad (3.40)$$

We performed some preliminary experiments with both methods. In our experience, the second method which uses local optimization is less computationally expensive and performs better.

Feature-based Registration

Most of the single modality images shown in this section can be efficiently and robustly registered using automatic feature matching methods such as those based on scale-invariant feature transform (SIFT) [44]. The real value of the method presented in this section is its ability to deal equally well with single and multi-modality registrations based on the content of the images. The method works well for images such as Fig. 3.16(d) where a small number of matching features can be found or Fig. 3.17 where due to the multi-modal nature of the images automatic feature-based matching, using SIFT for example, is not possible.

3.5 3D Registration

3.5.1 Estimating Rotation Parameters

The gradient intensity (3.13) is a 2D function of azimuth and zenith angles for 3D images which is invariant to translation and scale. Hence, we can find the rotational misalignment between the images by calculating MI over a 2D data-set for 3D images. This improves the efficiency in two ways, firstly the registration has to be solved for a smaller 3-parameter space and secondly the dimensionality of the problem is reduced as MI is being computed over a 2D data-set rather than directly on 3D images.

We start by computing a discrete version of $\Gamma(\phi, \theta)$ according to (3.13) for the fixed and moving images. We use an angular resolution of $2\delta\phi = 2\delta\theta = 1.0^\circ$, which results in GI matrices of 180×360 size regardless of the size of the original 3D volumes. To find the optimal rotation parameters, we apply the transformation directly to the GI matrix of the moving image. As illustrated by Fig. 3.20, the GI matrix is transformed from spherical coordinates to Cartesian coordinates, the Euler transformation $T(\alpha, \beta, \gamma)$ is applied and the result is transformed back to spherical coordinates. Note that, based on the definition of the gradient intensity, $\rho = 1$ for spherical/Cartesian conversions.

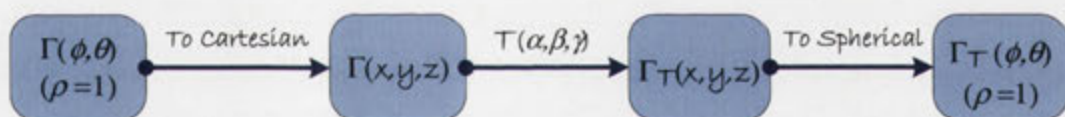


Figure 3.20: The process of transforming the GI matrix of a moving image by the three Euler angles α, β and γ .

The rotation parameters are then located by finding the maximum of the MI between the fixed GI matrix $\Gamma_{\mathcal{F}}$, and the transformed GI matrix $\Gamma_{\mathcal{M}(T)}$

$$[\hat{\alpha}, \hat{\beta}, \hat{\gamma}] = \underset{\alpha, \beta, \gamma}{\operatorname{argmin}} -\mathcal{S}_{\text{MI}}(\Gamma_{\mathcal{F}}; \Gamma_{\mathcal{M}(T)}). \quad (3.41)$$

The use of MI as a non-linear and statistical similarity measure is justified due to the multi-modal nature of the images which will result in non-linearly related GI matrices. Equation (3.41) can be solved as a 3-parameter optimization problem over a 2D data-set which is much easier to solve than the original 7-parameter (for isotropic similarity registration) or 6-parameter problem (for rigid registration) over the much larger 3D data-set. However, even though the parameter space is reduced, an exhaustive search strategy for 3 parameters is inefficient and we need

to use an optimization algorithm.

3.5.2 Soblex Optimization

The cost function in (3.41) is expected to have a global minimum where images are rotationally aligned. The cost function is smoother than 1D cost functions we have seen in Section 3.2. This is partly thanks to an increased number of samples used for MI computation. However, it may not be sufficiently smooth for a local optimization algorithm such as Powell or simplex that can be easily trapped by local minima and fail to converge to the global minimum. We propose a robust *semi-global* optimization method based on simplex and sampling of the parameter space with the Sobol quasi-random sequence [42], which we call the *Soblex* optimization.

The Soblex optimization is initially given a budget, in terms of time or the number of cost function calls. Within the initial budget, Soblex evaluates the cost function at points generated from a Sobol sequence. Once the given budget is exhausted, the algorithm initializes a standard simplex optimization using a simplex-shaped subspace, which is constructed from points with the lowest costs.

We say this method is a semi-global optimization as it comprises a global sampling phase and a subsequent local optimization phase. This combination improves the ability of the optimization method to locate the global minimum in the presence of local minima at a lower computational cost compared to global optimization methods such as DIRECT [25] or genetic optimization [26].

Using the Sobol sequence ensures that we can progressively sample the parameter space in a virtually uniform fashion. Intuitively, if the budget is large enough, the simplex subspace can sufficiently close in onto the global minimum to allow successful execution of the local optimization algorithm. The choice of simplex for the local optimization step is due to the fact that unlike most optimization methods that start from a single point in space, the simplex algorithm starts from a region of space that can be made arbitrarily close to the global minimum by increasing the Soblex budget. As can be appreciated using a single point for initialization increases the chance of a false start far from the global minimum. Whereas using multiple points for initialization, which is the case with simplex, makes a less biased start to the local optimization phase possible.

3.5.3 Estimating Translation Parameters

In general, we need to perform an optimization for the remaining set of parameters to complete the registration. However, rotationally aligned images are easier to

register. For example, multi-resolution methods are most useful for robust and quick recovery of translation misalignment between the images (in principle they offer no advantage for rotational misalignment or may even increase sensitivity to local minima [45]). Certain class of images may be amenable to faster registration methods when already rotationally aligned. For example, images of the head in brain imaging display a clear solid boundary for the skull. For this type of images, we can estimate translation and scale parameters by optimizing a 2D data-set as described below.

Let $\mathcal{F}(\mathbf{x})$ be the intensity of the fixed image at voxel \mathbf{x} , we call $\mathcal{F}_x(y, z)$ the reduced fixed image in direction of x . It is defined as a 2D image whose pixel intensities are calculated as the average of $\mathcal{F}(\mathbf{x})$ in the x -direction

$$\mathcal{F}_x(y, z) = \int_{-\infty}^{\infty} \mathcal{F}(\mathbf{x}) d\mathbf{x}. \quad (3.42)$$

If we reduce the fixed (\mathcal{F}) and transformed moving (\mathcal{M}') images along the three basis directions, we can estimate translation and scale parameters by optimizing a set of cost functions based on pairs of 2D images at considerably improved speed

$$[\hat{y}, \hat{z}, \hat{s}] = \underset{y, z}{\operatorname{argmin}} -\mathcal{S}_{\text{MI}}(\mathcal{F}_x; \mathcal{M}'_x), \quad (3.43)$$

$$[\hat{z}, \hat{x}, \hat{s}] = \underset{z, x}{\operatorname{argmin}} -\mathcal{S}_{\text{MI}}(\mathcal{F}_y; \mathcal{M}'_y), \quad (3.44)$$

$$[\hat{x}, \hat{y}, \hat{s}] = \underset{x, y}{\operatorname{argmin}} -\mathcal{S}_{\text{MI}}(\mathcal{F}_z; \mathcal{M}'_z). \quad (3.45)$$

For brain images, we use estimated rotation parameters to bring the images into rotational alignment, then reduce the volumes along principal axes and use Soblex to find the translation parameters. Note that the translation estimation shows some resistance to noise due to the averaging of the image layers. One could also include the isotropic scale parameter at this stage for similarity (rigid + scale) registration. While two out of the three possible optimizations would be sufficient to estimate all the parameters, in practice we run all three optimizations and average the results for improved robustness.

3.5.4 Final Optimization

We use estimations of rotation, translation and scale parameters in order to initialize an optimization algorithm based on Powell's multi-dimensional direction set. The final optimization round is performed using the standard PI-based MI cost function on the full 3D volumetric data to achieve sub-voxel accuracy. We take

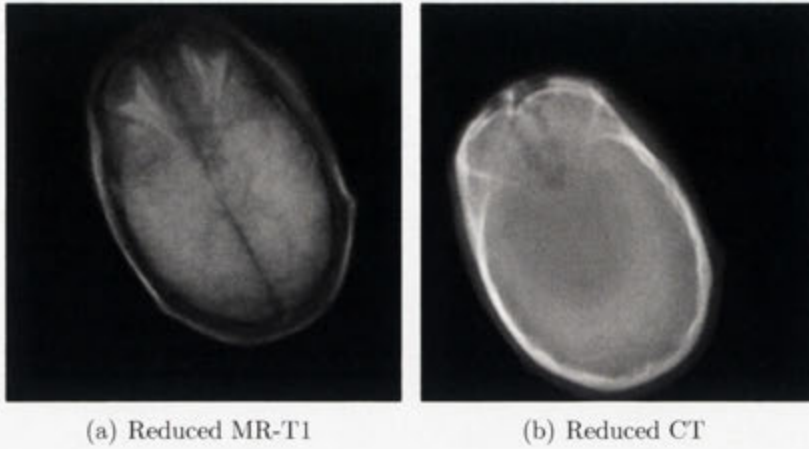


Figure 3.21: (a) MR-T1 and (b) CT images, rotationally aligned and averaged along the z axis. The 2D images are used to determine the x and y translation between the images. Note that the silhouettes are similar, which allows the MI-based optimization to quickly converge and return the translation parameters.

advantage of the fact that our optimization is initialized close to the final alignment by refraining from checking the perimeter excessively using the enhanced Powell implementation described in Section 3.2.3.

The choice of the final optimization method is somewhat arbitrary. We chose Powell, primarily because it does not require calculation of the cost function gradient, only needs one point for initialization, and could be more easily adapted to our finite resolution method.

3.5.5 Results

We evaluated the performance and efficiency of our method using various images from the *Retrospective Image Registration Evaluation* project (RIRE) database [46], where a gold standard for registration using fiducial markers was known. We considered CT to MR-T1, T2, PD and PET to MR-T1, T2, PD and MR-T2 to T1 registration. The images were rather low quality and low resolution (refer to Table 3.4), which made the registration more challenging and allowed us to experiment with our method under a more difficult condition.

The images in the RIRE database were brought into alignment using the gold transformation, then 100 random rigid transformations were applied to one of the images for each pair and our method was used for registration. The dynamic range of transformations were $[-25^\circ, 25^\circ]$ for rotation parameters and $[-32\text{ mm}, 32\text{ mm}]$ for translation parameters. The conventional PI-based method was used as the baseline for comparison. We used the standard implementation of Powell for the

Table 3.4: Resolution and size of images used in the experiments.

Image	Dimensions	Voxel Size (mm)
MR	$256 \times 256 \times 26$	$1.25 \times 1.25 \times 4.00$
CT	$512 \times 512 \times 29$	$0.65 \times 0.65 \times 4.00$
PET	$128 \times 128 \times 15$	$2.59 \times 2.59 \times 8.00$

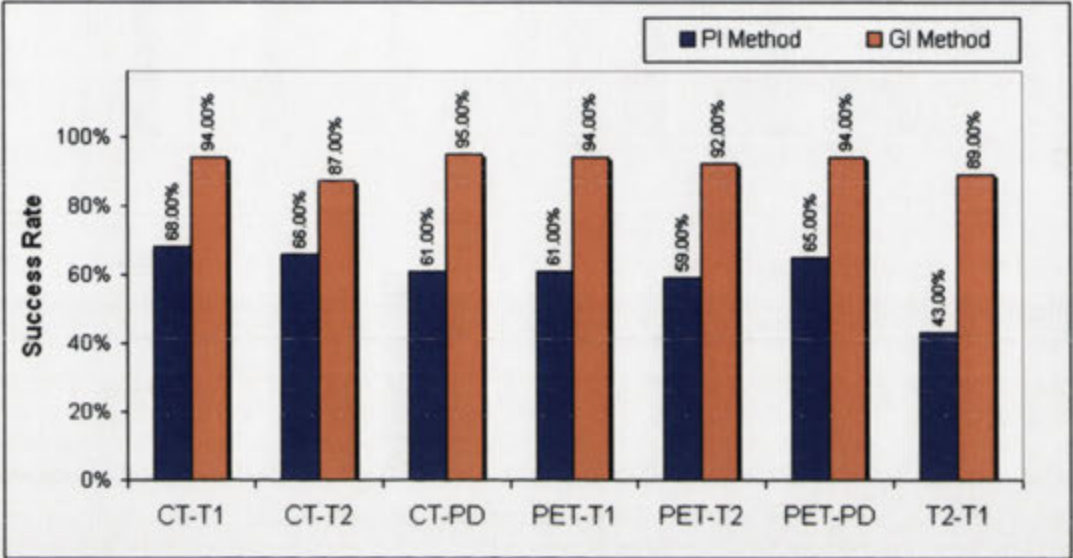


Figure 3.22: Superior performance of the GI-based method for various combinations of modalities.

conventional PI-based method, which was initialized from the origin of the parameter space. The registration errors were calculated by averaging the registration errors of 10,000 equally spaced points within the brain volume. Soblex optimizations were given an initial budget of 1000 cost function calls for rotation estimation and converged with an average of 1100 calls (including the initial budget). For translation estimation Soblex with an initial budget of 100 function calls was used and converged with an average of 150 iterations. For translations a much smaller initial budget can be used as the cost function is much smoother for the pixel intensities compared to the gradient intensities.

Performance

We declared a registration failed if the average error exceeded the diagonal voxel size of the moving image, which was approximately 8.80 mm for PET images and 4.40 mm for CT and MR images. Fig. 3.22 compares the success rate of our method

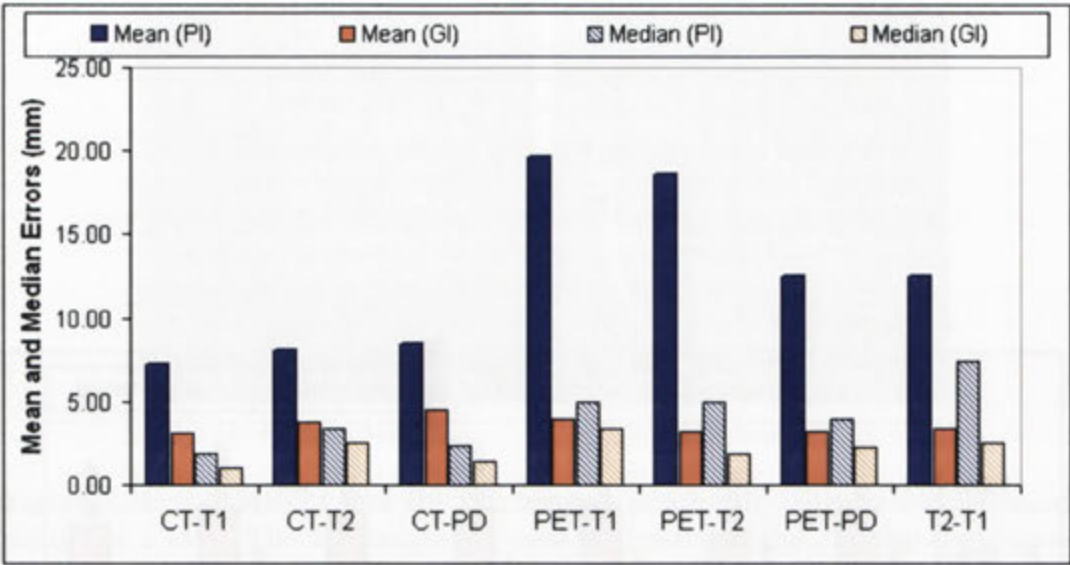


Figure 3.23: Improved accuracy of the GI-based method with a lower mean and median error is demonstrated. PI mean, GI mean, PI median and GI median errors are shown for each combination of modalities from left to right.

with the conventional method for various registration pairs and shows the superior performance of our method. Note that our method outperforms the PI-based method by a large margin, even if a lower threshold is chosen for registration errors. This can be better demonstrated by looking at the mean and median error graphs presented in the next section.

Accuracy

Fig. 3.23 shows average and median registration errors for each method. Registration errors are significantly reduced using the GI-method, due to the ability of the method to by-pass local minima. Also note that, median and mean errors are close for the GI method, which indicates the superior robustness of the method.

The improved statistical accuracy of our method compared to the standard PI-based method, as demonstrated in Fig. 3.23, is the result of improved robustness and success rate of our method, whereas the PI method fails to converge for large misalignments. However, we emphasize that where both methods converge, the accuracy of the standard method is similar to our method. This is because both methods use the same MI-based registration as the final stage.

Interpretation of errors should be treated with care. A common approach is to identify a number of volumes of interests (VOIs) and average the distance between centroids of VOIs, in registered images [1]. Obviously, error calculations will depend on the selection of VOIs and their distance from the center of transformation, as

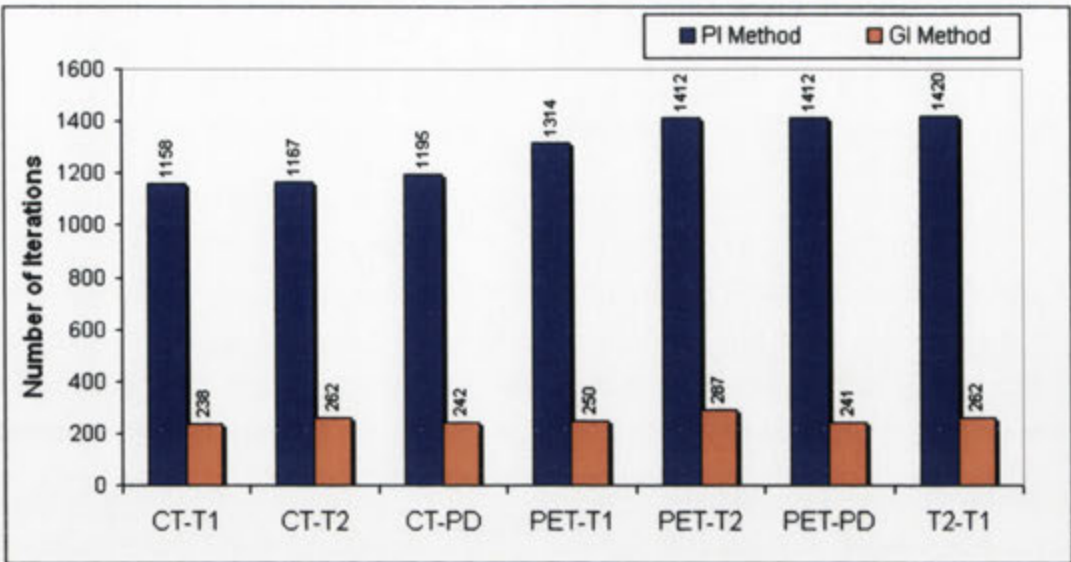


Figure 3.24: Superior efficiency of the GI-based method for various combination of modalities. The number of iterations are significantly reduced using the GI method. Computation times are proportional to the number of iterations.

well as accuracy of segmentation of VOI pairs. The advantage of this method is that the error is meaningful for anatomical or pathological details of interest. The statistical-based method, on the other hand, does not require segmentation and definition of VOIs. However, it may overestimate the errors by inclusion of points that may not be of interest. Hence, one should not directly compare error calculations obtained by different methods¹.

Efficiency

Both the GI and PI-based methods finally register the images using a Powell based optimization over pixel intensities. The computational cost of the methods are directly comparable at this stage. We included the computational cost of pre-processing required by the GI method by increasing the number of Powell iterations. The Soblex optimization stage of the GI method was approximately 1/100th of the cost of each iteration for the final Powell-based optimization or on average equal to 11 Powell optimizations. This has been factored in the end-results to allow a fair comparison. Another 15 iterations were added to account for gradient intensity calculations. Fig.3.24 compares registration efficiency of the GI and PI-based methods, and shows around 500% improvement as a result of using the GI

¹At the time of performing this set of experiments, we did not have access to VOI data for error measurements. In later experiments, such as those in Chapter 5, we used VOI-based error measurements.

method.

Powell Resolution

So far, we presented the results using a high resolution of $0.001\text{mm}/0.001^\circ$ for the modified Powell method. The computational efficiency of our method can be easily balanced against the required accuracy by selecting an appropriate resolution. To demonstrate this point, we now consider a medium and a low resolution of $0.01\text{mm}/0.01^\circ$ and $0.1\text{mm}/0.1^\circ$, respectively. The results in Fig. 3.25 for PET to T1 registration show that the computational efficiency can be considerably improved at the cost of moderate reduction in accuracy, if needed.

Robustness w.r.t. Noise

Gradient based methods are often more sensitive to noise. To demonstrate the robustness of the method w.r.t to noise, we added white Gaussian noise with $\sigma = 0.1$ to a PET image as shown in Fig. 3.26 and registered the image to MR-T1 100 times with random transformations applied to the PET image. The mean and median errors were 4.99 mm and 4.57 mm, up almost by only 1.00 mm compared to registration results without noise. The good results are to some extent due to the noise reduction feature of the uniform volume histogram (other noise resistant elements of our algorithm include the translation estimation process and the Gaussian gradient kernel itself).

3.5.6 Discussion

Gradient intensity is useful in determining the spatial orientation of 3D images and can be used to rotationally align the images, irrespective of their relative size and position. We experimented with various modalities of brain images and showed that our method is robust and performs well for all combinations, including the low resolution and conventionally challenging PET images.

We used an MI-based registration method in the final registration stage. However, the GI-based registration can be combined with other registration techniques and similarity measures. The registration performance using our method was on average 92%, up almost by 30% compared to the conventional PI-based method. The computational efficiency improved on average by 510% and median error was down from an average 4.13 mm to 2.14 mm across the whole range of our experiments comprised of 700 registrations for a relatively large range of misalignments.

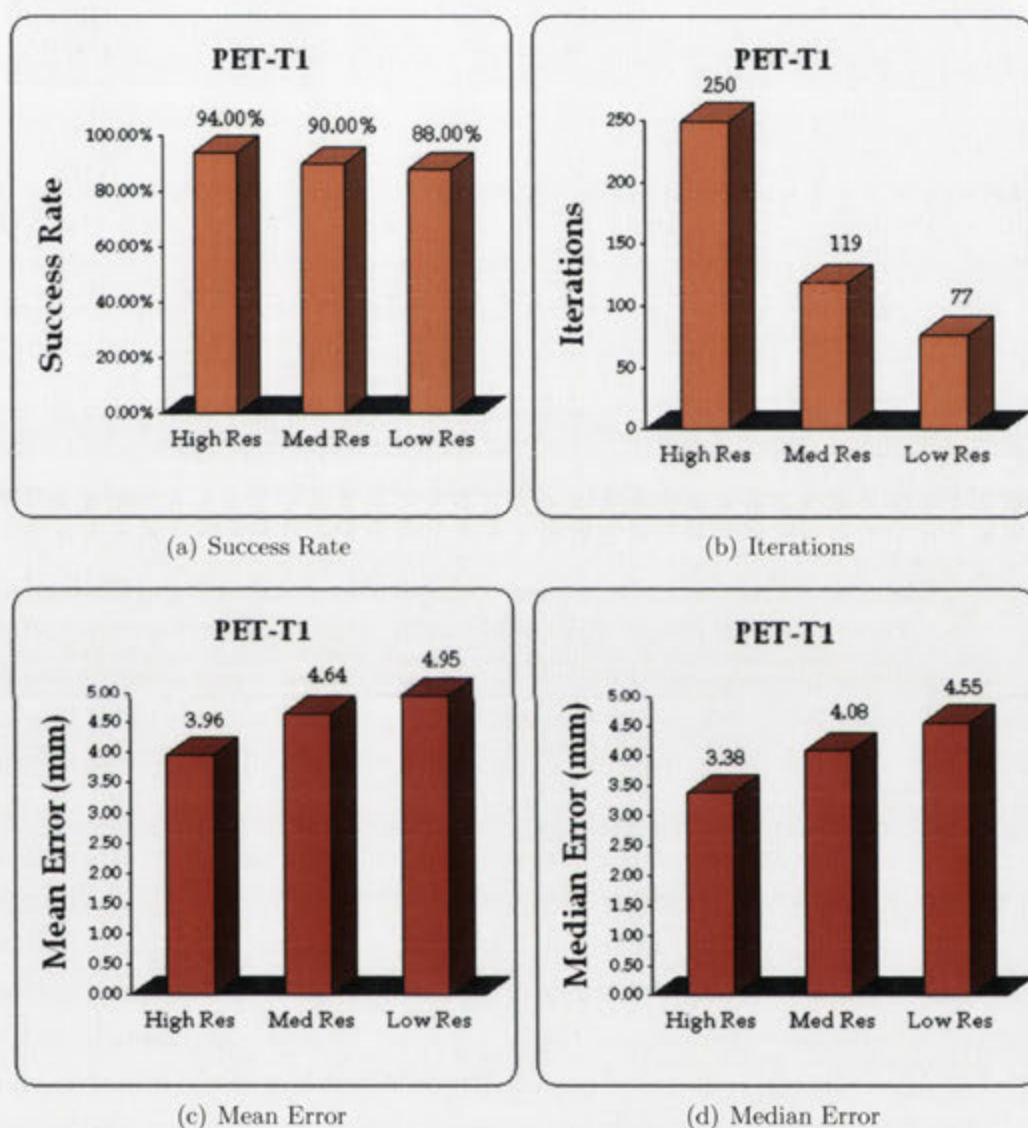


Figure 3.25: Reducing the resolution improves the computational efficiency, however it comes at the cost of a slight reduction in accuracy and success rate.

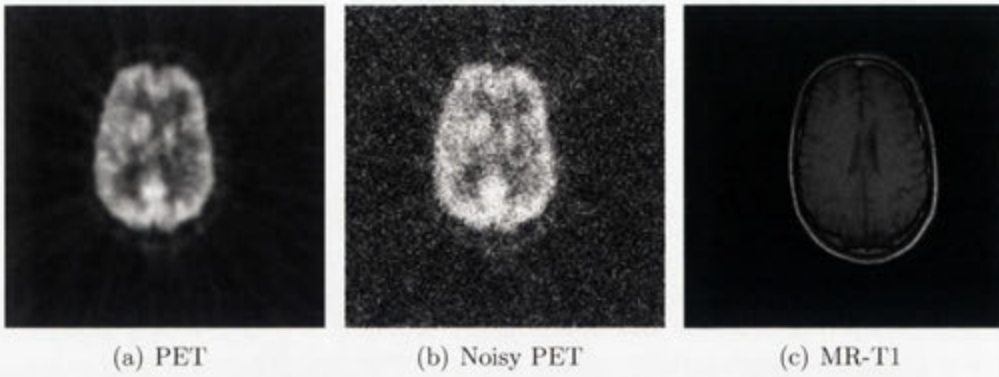


Figure 3.26: A PET volume (a) and its noisy version (b) registered to MR-T1 (c) in our experiments. The original PET volume itself has a low resolution and SNR.

In our experiments, we encountered a few cases where the final optimization algorithm had to recover from a completely incorrect initial estimate. This is not necessarily worse than starting from any other arbitrary position within the parameter space, such as the origin. However, the algorithm converges with performance and efficiency levels comparable to the PI-based method. Such cases can be further avoided and the performance can be improved by using a method that can estimate translation and scale independent of the rotation. Another avenue for getting better performance is to improve the GI-based cost function to obtain more accurate and more robust initial estimates. One such improvement is to consider a weight function for GI calculation to compensate for the non-linear size of the GI bins on the unit sphere. Another potential extension is to add a multi-resolution scheme for GI-based registration.

The idea of maximizing gradient intensity for alignment of images can be used for non-rigid registration. However, the method will no longer provide a computational advantage as the dimensionality of the problem cannot be reduced nor the optimization parameters can be estimated independently. In this capacity, mutual information of GI may be used to complement a PI-based cost function and to add spatial constraints to improve the robustness of registrations. The approach is along the line of combining gradient and intensity information as proposed in [34], except that our gradient cost function will be based on the GI.

Chapter 4

Registration on High Performance Computing Architectures

The advancements in development of multi-core and massively multiprocessing architectures in recent years holds great promise for interventional setups. In particular, massively multiprocessing graphics units with general purpose programming capabilities have emerged as front runners for low cost high performance processing. HPC, in the order of 1 TFLOPS, is available on commodity single-chip GPUs with power requirements not much greater than an office computer. Multi-GPU systems with up to 8 GPUs can be built in a single host and can provide a nominal processing capacity of 8 TFLOPS with less than 1500W power consumption under full load.

Hardware and architectural complexities in designing multi-core systems aside, perhaps as big a challenge is an overhaul of existing application design methodologies to allow efficient implementation on a range of massively multi-core architectures. As one quickly might find, direct adaptation of existing serial algorithms is more often than not neither possible due to hardware constraints nor computationally justified.

In this chapter, we look at early, recent, and state-of-the-art methods for registration of medical images on a range of HPC architectures including symmetric multiprocessing (SMP), massively multiprocessing (MMP) and architectures with distributed memory (DM) and non-uniform memory access (NUMA). We will define and describe concepts of interest in the context of image registration and high performance computing. Our main focus will be HPC-related aspects and we will highlight relevant issues as we explore the problem domain. This approach presents a fresh angle on the subject than previously investigated by the more general and classic reviews such as [2–4] in the literature. Section 4.1 and Section 4.2 are or-

ganized from the perspective of high performance and parallel computing with the registration problem embodied. This is meant to equip the reader with the knowledge to map a registration problem to a given computing architecture. Finally, we have endeavored to provide a comprehensive summary of existing contributions from various groups in Section 4.4.

4.1 Multi-CPU Implementations

4.1.1 Symmetric Multiprocessing

In symmetric multiprocessing (SMP) architectures multiple CPUs/cores have access to a single shared main memory. This makes parallelization of serial code relatively straightforward. The main methods for parallelization on SMP architectures are *POSIX threads* (pthreads) and *OpenMP* [47, 48]. The pthreads standard defines an application programming interface (API) for explicit instantiation, management and synchronization of multiple threads, whereas OpenMP mainly consists of a set of compiler directives (and a supporting API) that allows for implicit parallelization.

Most serial programs can be parallelized on SMP architectures with minimal modification. The ease with which parallelization can be achieved, especially with OpenMP, can be deceiving. We emphasize that one has to be prepared to reevaluate the approach to solving a problem on parallel systems and avoid the temptation of simply mapping a serial code to multiple threads. Use of *synchronization* primitives¹ should be limited to a minimum and alternative methods to achieve an outcome without synchronization should be investigated.

A good example of SMP parallelization of a registration algorithm is given by Rohlfing et al. [49]. They use pthreads to parallelize B-spline deformable registration on 64 CPUs. They exploit a combination of procedural (pre-computation, multi-resolution, adaptive activation of control points) and architectural elements (e.g. data partitioning) to optimize their method. While the hardware has been long superseded, their approach is still relevant today. We would not change much about their method except that they use synchronized reduction of partial joint histograms in MI computation phase by using the *mutex* lock. One can avoid the need for synchronization by dividing partial histograms and the resulting global histogram among the available threads. For N threads, this divides each partial

¹Synchronization refers to any mechanism for coordinating multiple threads or processes to complete a task. Examples of synchronization primitives include mutual exclusion (*mutex*), thread-join, and barrier. Atomic operations also involve implicit synchronization.

histogram into N equally sized non-overlapping regions. Each thread, then, computes part of the global histogram by summing values across corresponding regions of partial histograms. Since the regions are non-overlapping, the computations are guaranteed to be free of write-conflicts and no synchronization is required. We discuss this approach in more detail in Section 5.2.2.

4.1.2 Multiprocessing with Non-Uniform Memory Access

Efficient memory access is an important design consideration in multiprocessor systems with many cores where maintaining an efficient cache coherency on a single-shared-bus becomes less practical as the number of processors increases. Non-uniform memory access (NUMA) architectures try to alleviate the problem by dividing memory into multiple banks; each assigned to one processor. Processors have faster access to their local bank than remote banks attached to other processors.

Access to memory on remote banks can be several times slower than access to local memory. This is due to data traveling through a longer path and also transient access requests by other processors that may require the memory bus to be shared. Fig. 4.1 shows the schematic of a multiprocessor system with a NUMA architecture. An algorithm which is optimally designed for NUMA makes only infrequent attempts to access data on remote banks. A parallel application can theoretically achieve linear scalability with respect to memory throughput whenever proper distribution of memory to local banks is possible.

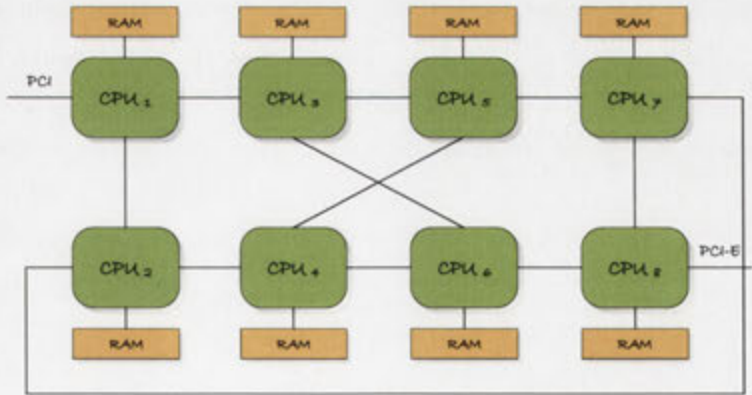


Figure 4.1: SunFire X4600 M2 schematic with 8 NUMA nodes. A CPU can access remote memory through a maximum of 3 hops.

Image registration can be efficiently implemented on NUMA architectures as shown in Fig. 4.2. Both the transform and measure computation can work on a spatial subset of the images. To achieve optimal performance, the fixed image F is

divided among the memory banks and the corresponding portion of the transformed moving image $M(T)$ will also be stored on the same memory bank. However, the path taken by the optimization algorithm cannot be determined a priori and the transformer will use different areas of M to create the local portion of $M(T)$ at each iteration. As such, each memory bank will need to receive a local copy of the moving image M during the initialization step. Given that the optimization algorithm will take several iterations to converge, this initial overhead is justified.

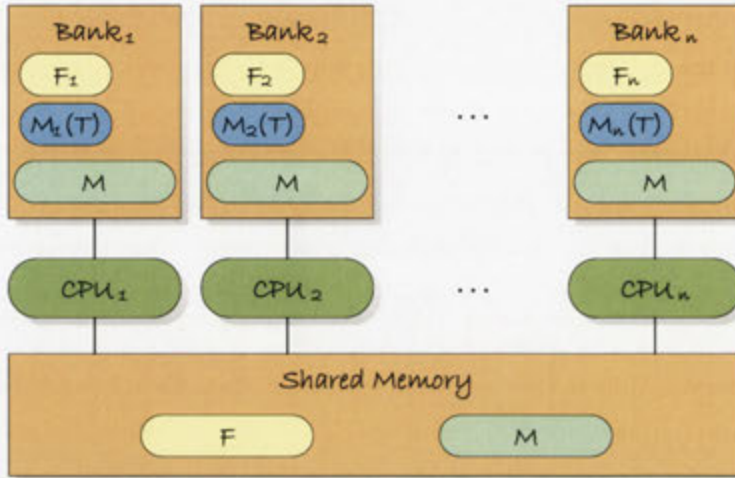


Figure 4.2: Partitioning of the data-set among multiple memory banks for improved access. The original data is loaded from a shared storage medium.

The distribution of resources to specific memory banks requires setting an appropriate memory and processor *affinity*². This is operating system dependent and will make the code less portable. The alternative is, of course, to be completely oblivious to the memory architecture and hope that the compiler and the operating system will make the right decisions. This may not be an entirely unreasonable strategy depending on the number of processors and whether a program is memory-bound or CPU-bound. However, as the number of available CPUs increases or for programs that are memory-intensive, it becomes more important to design an optimal memory access strategy.

4.1.3 Multiprocessing with Distributed Memory

Distributed memory (DM) architectures are characterized by lack of access to a global shared memory available to all processors. DM systems are typically built

²Processor affinity refers to explicit binding of a thread to a specific processor. Memory affinity is explicit allocation of data on a specific memory bank.

by clustering SMP or NUMA nodes. As such, in distributed architectures, subgroups of processors have access to shared memory.

From a programming standpoint, these systems are characterized by the need for explicit data distribution and interprocess communication. The former has to be built into the application design and the latter is most commonly achieved through the *message passing interface* (MPI) [50].

The model given for data distribution in NUMA Fig. 4.2 can be equally applied here. An early implementation is given by Butz and Thiran [27], where a Linux cluster was used to speed up MI-based registration for a global genetic optimizer. In [51], Ino et al. further partition the moving image in order to reduce the memory usage. This is motivated by the need to process large images in the order of $1024 \times 1024 \times 590$ voxels. Partitioning both images also reduces traffic on the network during initialization. This can be an important consideration as the number of nodes increases and the overhead of the initialization phase compared to the optimization phase can no longer be ignored. Partitioning the moving image requires a prior estimate of the range of transformation parameters to ensure that a large enough portion of the image is loaded for the transformer.

A variation is given by *distributed shared memory* (DSM) architectures, where a large virtual address space is made available to all processes across all nodes. DSM can only hide the mechanism of communication between processes and not the associated latency. We argue that if the end goal is to achieve the highest performance, little benefit can be drawn from the convenience of a DSM architecture and the program should be designed to be aware of the locality of data.

Wachowiak and Peters [28] develop MI-based registration for a DSM architecture. Their implementation does not take memory locality into account but they use DIRECT and MDS parallel optimization methods to their advantage. This coarse-grained parallelization results in lower communication-to-computation overhead.

As some authors have pointed out [52], a major benefit of DM clusters is their lower cost compared to many-core SMPs or DSM systems.

4.2 Accelerator Implementations

4.2.1 Cell Broadband Engine

Cell/BE is an asymmetric heterogeneous multi-core processor with a distributed memory architecture. It comprises a general purpose PowerPC core known as a PPE and eight specialized vector processing cores known as SPEs. Each SPE is

equipped with a 4-way SIMD engine and has its own small (un-cached) memory known as the local storage³.

Optimal implementation of registration algorithms on Cell/BE architectures involves task-level parallelization, data partitioning, and vectorization of the code for the SPEs' SIMD engine. It also involves handling the transfer of data between the system memory and SPEs' local storage. The results by Ohara et al. [53, 54] and Rohrer and Gong [55] provide good insight into challenges involved in designing registration on this architecture for collinear and deformable registration, respectively.

4.2.2 Field Programmable Gate Array (FPGA)

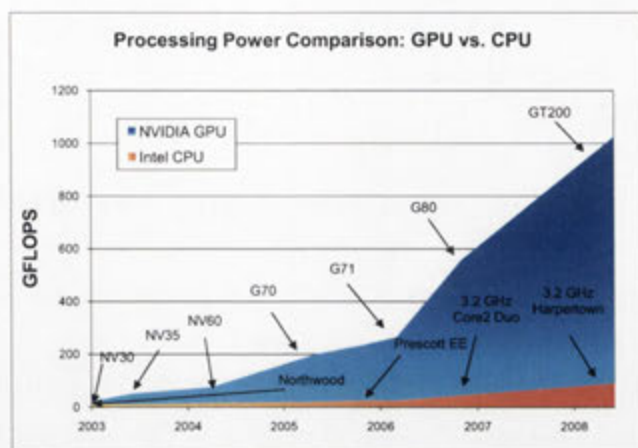
A custom FPGA accelerator prototype for MI-based rigid registration is given by Castro-Pareja et al. in [11]. They argue that a major bottleneck in MI computation using Collignon's method [15] is partial volume (PV) interpolation and that it is memory-bound. They improve performance by parallelizing access to memory and assigning independent memory controllers and types of memory for storage and access to the fixed image, the moving image, and the joint histogram. A *cubic* addressing scheme is used for the moving image to speed up the interpolation. This is similar to caching available in GPUs for access to texture memory. An enhanced version of [11] is presented in [56] and a multi-rigid version with volume subdivisions is given by Dandekar [57].

FPGAs allow one to design customized hardware for specific registration tasks. However, they provide less flexibility compared to software-based implementations. With flexible general purpose programming capabilities of modern GPUs, it is doubtful if FPGA-based implementations present any real benefit in this area.

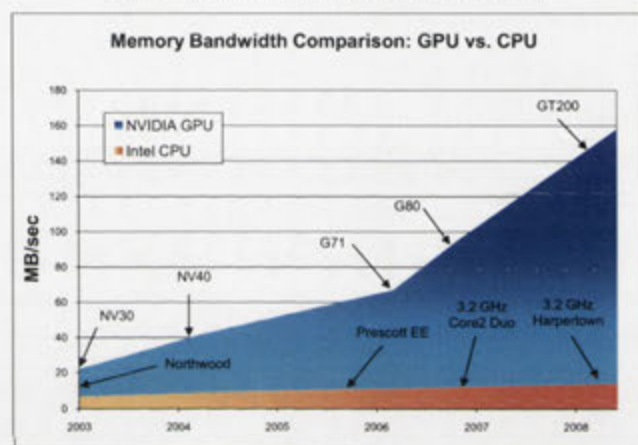
4.2.3 Graphics Processing Unit (GPU)

The majority of recent research in multi-core adaptation of registration algorithms has been focused on GPUs [58–64]. The interest in GPUs stems from continued improvement in computational power and increase in memory bandwidth that has consistently outperformed Moore's prediction [65] in recent years (Fig. 4.3). At the same time, a competitive gaming market has driven the GPU prices down to the point that GPUs provide the highest computational performance per dollar of any HPC architecture.

³Local storage is only 256KB in current generation of hardware and is shared between data and kernel instructions.



(a) Processing power of GPU vs. CPU.



(b) Memory bandwidth of GPU vs. CPU.

Figure 4.3: Rapid increase in GPU's processing power and memory bandwidth in recent years (source of data: NVIDIA [66]).

Earlier work in this area (mainly prior to 2007) [9, 67–73] involved devising methods to map the registration problem onto a *graphics pipeline* which was not specifically designed for general purpose computing. The GPU landscape has since gone through a seismic change with the introduction of native general purpose computing capabilities in late 2006. The GPU registration literature prior to 2007 has been superseded from both hardware and software perspectives.

Despite improvements in programming model and available tool-sets, GPU implementations remain more challenging than multi-core CPU implementations. However, the lower cost and achievable performance gains make GPUs an important platform for HPC in many scientific applications. In the remaining of this chapter we will focus on the latest technology and algorithm design for general purpose computing on GPUs. This will serve as a precursor to Chapter 5 where

we discuss a number of histogram computation, MI computation, and registration methods designed for the massively multiprocessing architecture of the GPU.

4.3 Massively Multiprocessing on GPUs

The modern software platforms for general purpose programming on the GPU are currently NVIDIA's CUDA [66] and AMD/ATI's Brook+ [74]. These platforms are vendor-specific, however OpenCL compliant implementations that provide hardware-independence are being gradually released by the vendors. None of the papers we considered developed methods for ATI Brook+. It appears that the research community has almost exclusively adopted CUDA as their preferred GPU platform. This is likely to change with wider support for OpenCL in non-GPU architectures such as IBM's Cell/BE and Intel's Larrabee.

Modern GPUs extend the *single instruction multiple data* (SIMD) paradigm to a *single instruction multiple threads* architecture (SIMT). SIMT provides more flexibility by parallelism for (almost) independent threads as well as data-parallel code. GPUs achieve their computational performance by dedicating more transistors to their arithmetic logic units (ALUs) for data processing, at the expense of reduced flow control and data caching. They extend the conventional *thread*-level parallelism by introducing two additional layers of parallelism in the form of closely knit groups of threads known as *warps* or *wavefronts*, and groups of warps/wavefronts known as *thread blocks* or simply *blocks*. Warps are significant since they define the unit of flow control in a GPU. Threads in a warp are bound to execute the same instruction (on different data). Diverging paths of execution for threads in a warp result in serial execution of all paths. Hence, an important consideration in adapting parallel code to GPU architecture is minimizing diversion in warps. This can be achieved by designing *warp-aware* algorithms and reorganizing data to optimize flow control. An example of such an approach is given in [63].

Limited data caching is another important consideration in designing programs for the GPU. To achieve the best performance one needs to understand the hardware architecture and its various memory and caching models. Optimum use of memory such as *coalesced* transfers may speed up an application by an order of magnitude. This level of flexibility is typically available with lower level APIs and runtime SDKs such as CUDA (NVIDIA) [66] and CAL (ATI/AMD) [74]. Programs developed with a lower level API lack portability and need to be maintained as the hardware evolves. Abstraction layers such as OpenCL and Brook+ avoid these issues by hiding memory management details from the developer. However, better

portability may come at the cost of sub-optimal performance.

Different MI computation methods on the GPU have been reported in the literature. Shams et al. compute MI by computing joint histograms on the GPU in [59, 63, 75]. A main finding is that for different sized histograms (number of bins used for MI computation), the optimal algorithm differs. For bin ranges typical in MI computation (100×100 and above) an efficient histogram computation algorithm specifically designed for massively multiprocessing architectures is presented in [63]. The paper describes a new method for histogram computation (*sort and count*) that removes the need for synchronization or atomic operations, based on sorting chunks of data with a parallel sort algorithm such as *bitonic* sort. Lin and Medioni [60] report an adaption of Viola's MI computation approach [16]. Their method approximates the joint pmf by stochastic sampling of the image intensities and using Parzen windowing. This method lends itself well to parallelization on the GPU, reduces the computational burden of transformations by only using a subset of image data, and provides analytic equations for computation of MI derivatives. However, sparse sampling of the data-set may compromise accuracy of the registration [68]. A sampling method specifically designed for the GPU is given by Shams and Barnes [59]. This method samples the bin space for computing histograms rather than the intensity space. The method improves performance of computations and is subject to the same trade off between performance and accuracy. We note that a majority of researchers use direct computation of the histogram [4]. Finally, feature-based registration of large 2D data-sets (in the order of $16K \times 16K - 23K \times 62K$) using a polynomial transformation and the computation of CC similarity measure on the GPU is reported by Ruiz et al. [64].

4.3.1 An Overview of CUDA

CUDA is a parallel computing architecture by NVIDIA that can be used to offload data-parallel and compute-intensive tasks to NVIDIA GPUs beginning with the 8 series since late 2006. It consists of an entire platform for development and execution of general purpose programs on the GPU comprising the device driver, programming APIs, C for CUDA compiler, and a *profiler*. In this section, we provide an overview of the terminology, main features, and limitations of CUDA. We have summarized the main concepts in Table 4.1 for quick reference. More information can be found in [66]. A reader who is familiar with CUDA may skip this section.

Execution Model

In CUDA, the computation is distributed in a *grid of thread blocks*. All blocks contain the same number of threads that execute a program on the *device*⁴, known as the *kernel*. Each block is identified by a two-dimensional block ID and each thread within a block can be identified by an up to three-dimensional ID for easy indexing of the data being processed. The block and grid dimensions, which are collectively known as the *execution configuration*, can be set at run-time and are typically based on the size and dimensions of the data to be processed.

It is useful to think of a grid as a logical representation of the GPU itself, a block as a logical representation of a multi-core processor of the GPU and a thread as a logical representation of a processor core in a multiprocessor. Blocks are time-sliced onto multiprocessors. Each block is always executed by the same multiprocessor. Threads within a block are grouped into *warps*. At any one time a multiprocessor executes a single warp. All threads of a warp execute the same instruction but operate on different data. As such, a warp represents the atomic unit of program flow on the device. This means that if threads of a warp take diverging path of execution within a kernel, they will have to be serialized. This may happen when a data-dependent branch is executed by a warp. The warp will execute all visited branches and for every visited branch disables those threads that are not supposed to be executed. From a programming perspective, warps are a significant improvement over *vector processing architectures* where data has to be explicitly loaded into a vector and any potential divergence to be handled manually by the programmer. Full efficiency can be achieved only when all threads within a warp can remain active. An important consideration in designing code for the GPU is to organize the code and data access patterns such that diverging branches for the threads of a warp are minimized.

Another important consideration in GPU-based applications is designing an optimal memory mapping scheme. A GPU has several memory spaces, each with differing levels of functionality in terms of size, latency, throughput, and caching mechanism. The various types of memory include *global*, *texture*, *constant*, *shared*, and *local* memory.

Memory Model

The device's DRAM, the *global memory*, is un-cached. Access to global memory has a high latency (in the order of 400-600 clock cycles) [66], which makes reading

⁴We use the terms *device* and the GPU, and *host* and the CPU interchangeably.

from and writing to the global memory particularly expensive. Fortunately, much of the latency can be hidden by the thread scheduler providing that there are other threads that can be placed for execution while waiting for the data transfer to complete. To fully hide the latency, one typically needs to ensure that the kernel has a high ratio of arithmetic operations for every memory access operation and should run the kernel with an execution configuration that allows for hundreds of blocks and several hundred threads per block.

The throughput of global memory access is also dependent on the access pattern. When certain requirements are met by threads in a warp, access to global memory by multiple threads can be combined into a single transaction for contiguous memory locations. This is known as memory *coalescing*. Non-coalesced memory access can severely affect the performance of an application and should be avoided where possible. Coalescing global memory access is perhaps the single most important consideration in optimizing CUDA code [76]. It may even be worthwhile to reorganize data prior to execution of a kernel in order to ensure coalesced access. The exact requirements for memory coalescing differ for different generations of GPUs and we refer the reader to [76] for a detailed discussion.

The data is transferred between the host and the device via the *direct memory access* (DMA), however, transfers within the device memory are much faster. To give the reader an idea, device to device transfers on GTX 8800 and GTX 280 are around 80 GB/s and 140 GB/s, respectively, whereas, host to device transfers can be around 2–3 GB/s. As a general rule, host to device memory transfers should be minimized whenever possible. In a complex application with several processing phases, data should be initially loaded onto the GPU and kept there for as long as possible. With a GTX 8800, for example, there is a latency of around 20 μ s for memory transfers [77]. The transfer rate is lower for smaller data-sets, for example the transfer rate for a 1 KB data-set is only 0.04 GB/s, whereas for 1 MB and 100 MB the transfer rate increases to 2.70 GB/s and 3.10 GB/s respectively. Therefore, several smaller data transfers should be combined into a single transfer, where possible.

Areas of the global memory can be mapped as read-only *texture memory*. The texture memory is cached and also optimized for 2D and 3D indexing. This is particularly useful for image processing applications that frequently access adjacent data elements in a rectangular or cubic grid. Textures also provide hardware accelerated support for linear interpolation of adjacent data elements in a grid.

Shared memory is a small (16 KB) on-chip memory located within each processing core. It is visible to all threads within a block and acts as the primary

mechanism for inter-thread data cooperation. It is also meant to compensate for the lack of transparent caching. It is divided into a number of banks that can be accessed simultaneously (given a suitable access pattern that avoids bank conflicts). Shared memory latency can be up to 100 times lower than global memory latency [76]. It is typically used to pre-load small chunks of data, frequently accessed by several threads, from global memory and to store intermediate computation results. The efficiency of a kernel can be improved by taking advantage of parallel access to shared memory and by avoiding bank conflicts. In practice, the size of shared memory is too small for many applications. This is a limitation that has to be overcome by the programmer and at the expense of a more complicated kernel code and reduced performance.

Constant memory is a small 64 KB of read-only cached memory. It is visible to all threads and as the name suggests is used to store numeric constants used by all threads such as filter coefficients, transformation matrices or parameter values in parameterized equations.

The last type of memory that we briefly mention is confusingly named *local memory*. Local memory is in fact off-chip with an access latency similar to global memory. The 'local' designation is apparently related to its scope which is local to a thread. It is used by the compiler for allocation and storage of automatic variables such as large structures and arrays that would otherwise occupy too many registers. Local memory makes execution of complex kernels possible. However, performance-wise it is almost always bad news if the compiler is forced to resort to using the local memory.

We conclude the discussion on memory by highlighting an important limitation of the current GPU memory architecture: lack of support for error correcting code (ECC). This is not an issue for graphics applications for which a GPU is primarily designed. An infrequent flipping of a single memory bit (known as a *soft error*) may affect the color of a pixel or cause a transient glitch. In a computer game this will largely go unnoticed. The same cannot be said for general computation on the GPU. Today, ECC is not optional for serious scientific research and in computationally sensitive applications such as computational finance. [78] reports a failure rate of 1% during the initial launch of Stanford's distributed protein folding application, Folding@Home. If this figure is to be believed⁵, the problem is indeed serious and one has to consider redundant computation options such as discussed

⁵The high failure rate of Folding@Home has been attributed to often over-clocked and non-standard configurations employed by users who donate their computational resources to the project. Even so, we find it hard to accept that this high rate of failure can be entirely blamed on soft errors.

in [78] for mission-critical GPU-based applications.

Synchronization Support

The higher processing power of the GPU compared to the standard CPU, comes at the cost of reduced data caching and flow control logic as more transistors have to be devoted to data processing. This imposes certain limitations in terms of how an application may access memory and implement flow control. As a result, implementation of certain algorithms (even trivial ones) on the GPU may be difficult or may not be computationally justified. In particular, CUDA devices with *compute capability 1.0* (such as GTX 8800) do not support *atomic* operations. GPUs with compute capability 1.1 support some atomic operations on the global memory. Newer GPUs with compute capability 1.3 (such as GTX 280) support some atomic operations in the shared memory. However, existing GPUs still lack other synchronization primitives such as *critical section* and *mutual exclusion* (mutex). The only universally supported synchronization primitive is the *thread join* which only works among the threads of the same *thread block*.

In designing an algorithm for a massively multiprocessing architecture, regardless of the availability of synchronization and atomic features, one should minimize dependence on synchronization among threads, as it essentially causes parallel processes to become serialized and reduces performance.

A Typical CUDA Program

A typical CUDA implementation consists of the following stages:

1. Allocate memory on the device.
2. Initialize device memory if required.
3. Transfer data from the host to the device.
4. Determine the execution configuration.
5. Execute kernel(s). The result is stored in the device memory.
6. Transfer data from the device to the host.

With the exception of step 5, everything is an overhead. Naturally, a GPU implementation is justified only when the computational benefit of step 5 outweighs the overhead. In complex, iterative or multi-phase algorithms the efficiency can be greatly improved if all the computation can be performed on the GPU, so that

Table 4.1: A quick reference of CUDA related terminology

Terminology	Description
Device	a generic name for an auxiliary computational unit (e.g. a GPU) acting as a coprocessor in a heterogeneous system.
Host	the primary computational unit of a heterogeneous system.
Kernel	an extension to a standard C function that defines the computational work-load performed by a single thread.
Thread	the atomic unit of program execution with its own set of registers and local memory.
Warp	the atomic unit of program flow comprising a number of tightly coupled threads that are bound to execute the same set of instructions at any one time.
Block	a group of threads with access to a shared memory space run on a single multiprocessor.
Grid	the collection of all thread blocks which represents the entire execution context on a single GPU core.
Constant Memory	a small read-only cached memory accessible by all threads in a grid, typically used to store program and computational constants.
Global Memory	the primary device memory accessible by all threads in a grid with no caching support.
Local Memory	a un-cached memory space private to each thread used to offload variables deemed to use too much register space by the compiler.
Shared Memory	a small ultra-fast memory accessible by each multiprocessor and the threads within a block, typically used to store variables frequently accessed by multiple threads.
Texture Memory	a large read-only memory with limited caching accessible by all threads in a grid, typically used for locally coherent access patterns with 1D, 2D and 3D indexing.

step 5 can be run several times without the need for unnecessary data transfers between the device and the host.

4.4 Summary of the Literature

We have summarized existing contributions in high performance computation of registration methods in Table 4.2, Table 4.3, and Table 4.4. The tables serve as quick references to an array of methods on various platforms and by different groups.

Researchers have used various methods to present their performance results. All groups report at least the speedup results compared to a single-core CPU implementation. When inter-architecture comparisons are drawn, it is not always clear how well the CPU implementation has been optimized, if the streaming SIMD extensions (SSE) instruction set has been used, whether the code has been compiled as 64- or 32-bit, or if 64- or 32-bit floating point operations have been used. For these reasons, speedup results should be interpreted with caution, more so when the reported speedups are in the order of a hundred times or more.

Most groups report their speedups for the entire registration algorithm and for specific data-sets. Comparison of different results is further complicated as authors may have implemented a multi-resolution scheme to further speed up the process and used different convergence criteria. We have reported/estimated the results for the finest resolution, whenever possible. The execution time is an almost linear function of the number of iterations of the optimization algorithm. Convergence criteria are most commonly based on the value of the measure and its relative improvement in a given step of the optimization. A less common approach is to set a fixed number of iterations as the convergence criterion. We call the former strategy *dynamic convergence* and the latter *static convergence*. Lack of associativity for floating point operations have the inevitable consequence that the same optimization algorithm operating on the same data-set converges with different number of iterations on different architectures when dynamic convergence is employed. Even on the same architecture, compiler optimization of floating point operations results in variations. Unless experiments are performed on a large set of images, this skews the performance results one way or the other.

We have given normalized *performance*⁶ results where possible. The purpose of

⁶The word ‘performance’ is ambiguous in the context of registration. It is sometimes used to refer to the degree of success for a registration algorithm based on accuracy of the registration results. In this section, we use ‘performance’ in its computational capacity refereing to execution efficiency of the registration algorithm.

normalizing the reported results is to give the reader an indication of the speedups expected from a method without dependence on the size of images involved, convergence criteria, use of a multi-resolution scheme, and to some extent the type of optimization algorithm. Normalized results are given in terms of average execution time in milliseconds for a single iteration of the optimization algorithm and for processing 1,000,000 voxel pairs (ms/MVoxel/itr).

Many authors have used gradient descent as their optimization algorithm, largely due to its simple structure and ease of implementation. Once the gradient is computed, the choices include taking a single step in a direction opposite to the gradient where the step size may be adjusted over time, or use of a line minimization algorithm such as Brent's [22]. Line minimization usually involves several computations of the cost function alone without its derivatives.

When comparing results it is important to identify which variation of the gradient descent is used. We have come across four different implementations.

- Type A: Closed-form differentiation with a single step
- Type B: Closed-form differentiation with line minimization
- Type C: Numerical differentiation with a single step
- Type D: Numerical differentiation with line minimization.

Most authors exclude initialization time, including disk IO and loading data from host memory to GPU memory. This is a reasonable practice since initialization time is typically a small fraction of the registration task. Initialization occurs at the beginning of the registration algorithm whereas the optimization loop is executed several times.

Some of the information presented in the following tables were not immediately available in the original manuscripts and were provided by the authors of the respective papers.

Table 4.2: Summary of high performance image registration methods in the literature on CPU-based architectures

	Transform	Meas.	Optimizer	Hardware	Perf. ¹	Group	
Collinear	Simil.	NML	Powell	2 × Sun Ent. 5000 (2 × 8 UltraSparc I 167MHz)	-	Warfield 98 [79]	1
	Affine	MI	Genetic	PC Cluster (10 × 2 Pentium III 550MHz)	-	Butz 01 [27]	2
	Rigid	LLC ³	?	PC Cluster (10 × 2 Pentium III 933MHz)	-	Ourselin 02 [52]	3
	Rigid	MI, NMI	DIRECT, MDS	SGI Altix 3000 (20 Itanium II 1.3 GHz)	-	Wachowiak 06 [28]	4
	Rigid	MI	Powell	Sun SPARC T5120 (8 × UltraSPARC T2 1.2GHz)	47.7	Shams ² 09	5
	Rigid	MI	Powell	Intel Q6600 (Pentium Core 2 Quad 2.4GHz, 4 cores)	15.8	Shams ² 09	6
	Rigid	MI	Powell	Intel Core i7 920 (Quad 2.66GHz, 8 threads)	13.2	Shams ² 09	7
	Rigid	MI	Powell	SunFire X4600 M2 (8 × 2 Opteron 2.6GHz)	10.5	Shams ² 09	8
Def.	B-spline	NMI	Grad. desc. (D)	SGI Origin 3800 (128 MPIS 12K)	-	Rohlfing 03 [49]	9
	B-spline	NMI	Grad. desc. (D)	PC Cluster (64 × 2 Pentium III 1GHz)	-	Ino 05 [51]	10

¹ : Normalized performance in milliseconds per mega voxel per iteration (ms/MVoxel/itr). ² : Previously unpublished result. ³ : The method is based on block matching with local linear correlation measure (LLC).

Table 4.3: Summary of high performance image registration methods in the literature on accelerator technologies

	Transform	Meas.	Optimizer	Hardware	Perf. ¹	Group	
Collinear	Rigid	MI	N/A	FPGA (2 × Altera 1K100 80MHz)	101	Pareja 03 [11]	11
	Rigid	MI	N/A	FPGA (1 × Altera EP1S40 200MHz)	20.0	Pareja 04 [56]	12
	Affine	MI	Grad. desc.	QS20 (2 × Cell/BE.: 2 × 1 PPE & 8 SPEs)	98.8	Ohara ² 07 [54]	13
Def.	Multi-rigid	MI	Simplex	FPGA (1 × Altera EP2S180 200MHz)	13.4	Dandekar ² 07 [57]	14
	B-spline	MI	Grad. desc.	QS20 (2 × Cell/BE.: 2 × 1 PPE & 8 SPEs)	66.9	Rohrer 08 [55]	15

¹ : Normalized performance in milliseconds per mega voxel per iteration (ms/MVoxel/itr). ² : Additional information provided by the authors used to complete the table or to compute normalized performance results.

Table 4.4: Summary of high performance image registration methods in the literature on GPU-based architectures

	Transform	Meas.	Optimizer	Hardware	Perf. ¹	Group	
Collinear	Rigid	SSD	Simplex	GeForce 6800	98.0	Köhn 06 [80]	16
	Rigid	SSD	Grad. desc. (B)	GeForce 6800	858	Köhn ³ 06 [80]	17
	Rigid	GC	?	Quadro FX 1400, FX 3400, GTX 7800	-	Ino 06 [69]	18
	Rigid	Various ⁴	Custom	GeForce 6800 GT	-	Khamene ³ 06 [68]	19
	Rigid	Various ⁴	ARS + BN	GeForce 7800 GS	-	Kubias 08 [73]	20
	Rigid	MI	Simplex	GTX 8800 (16 MP/ 128 cores)	6.17	Shams 07 [59]	21
	Rigid	SSD	Simplex	GTX 8800 (16 MP/ 128 cores)	6.05	Plishker ³ 08 [61]	22
	Affine	MI	Grad. desc. (A)	GTX 8800 (16 MP/ 128 cores)	-	Lin 08 [60]	23
	Rigid	MI	Powell	GTX 280 (30 MP/ 240 cores)	4.06	Shams 09 [63]	24
Deformable	Bezier	MI	Powell	GeForce3 64MB	-	Soza 02 [9]	25
	Non-par.	SSD	Grad. desc.	GeForce FX 5800 Ultra	-	Strzodka 04 [67]	26
	Non-par.	SSD	Grad. desc. (B)	GeForce 6800	465	Köhn ³ 06 [80]	27
	Non-par.	MI + KL	Grad. desc. (C)	GTX 7800	2860	Vetter ³ 07 [70]	28
	Non-par.	MI + KL	Grad. desc. (C)	GTX 8800 Ultra (16 MP/128 cores)	324	Fan ³ 08 [71]	29
	Demons	SSD	Iterative	Quadro FX 1400	1050	Courty 07 [72]	30
	Demons	SSD	Iterative	GTS 8800 (12 MP/96 cores)	11.7	Sharp ³ 07 [58]	31
	Demons	CC	Iterative	Quadro FX 5600 (16 MP/128 cores)	9.25	Özçelik 08 [62]	32
	B-spline	SSD	Grad. desc. (C)	GTX 8800 (16 MP/ 128 cores)	3710	Plishker ³ 08 [61]	33
	B-spline	SSD	Grad. desc. (D)	GTX 295 (30 MP/ 128 cores)	573	Shams ² 09	34
	B-spline	NMI	Grad. desc. (D)	GTX 295 (30 MP/ 128 cores)	1699	Shams ² 09	35
	Polynom.	MI	Exhaustive	Quadro FX 5600 (16 MP/128 cores)	-	Ruiz 09 [64]	36

¹ : Normalized performance in milliseconds per mega voxel per iteration (ms/MVoxel/itr). ² : Previously unpublished result. ³ : Additional information provided by the authors used to complete the table or to compute normalized performance results. ⁴ : Various measures were considered including SSD, CC, and GC.

Chapter 5

Registration on the GPU

As discussed in Chapter 2, image-based registration typically consists of several iterations of some optimization algorithm with the aim to minimize a suitable cost function subject to an optional smoothness criteria

$$T_{opt} = \underset{T}{\operatorname{argmin}} -\mathcal{S}(F; M(T)) + \mathcal{L}(T), \quad (5.1)$$

where \mathcal{S} is the similarity function to be maximized, \mathcal{L} is the smoothness term, T is a transformation operator, and F and $M(T)$ are the *fixed* and transformed *moving* images, respectively. Each iteration of the optimization involves transformation of the moving image and computation of the cost function against the fixed image. For ease of reference, we include Fig. 2.1 here again, where the major components of a general registration solver are depicted.

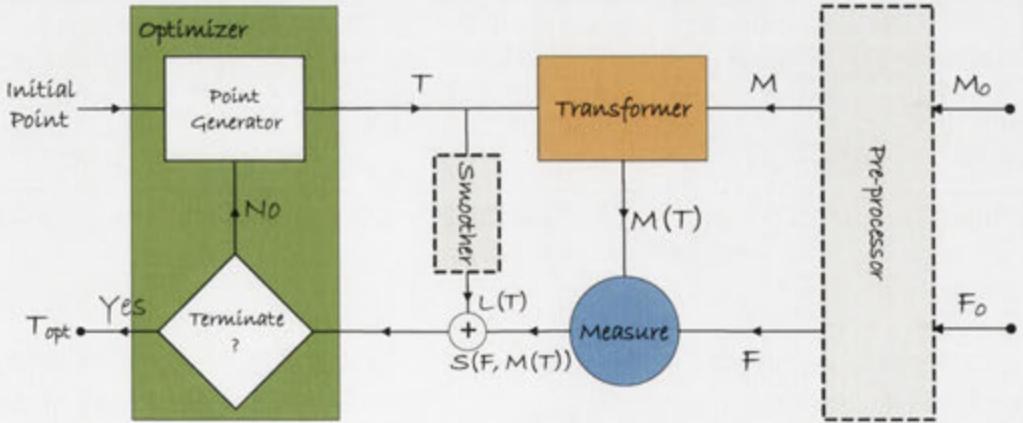


Figure 5.1: A general registration solver and its main components where F , M , and $M(T)$ are fixed, moving and transformed moving images, respectively.

In this chapter, we are concerned with development of efficient algorithms for the transformer and measure components on the GPU. These components represent

the main bottlenecks for an iterative registration, as depicted above. We would like to clarify, early on, that there is no computational benefit in implementing the optimizer on the GPU as the time spent in an optimizer outside of the cost function is typically less than 1%.

In the course of this chapter, we will demonstrate that GPU implementation of a serial algorithm is much more than a mere port of an existing code to a new platform. We argue that to maximize the benefits of moving to a massively multiprocessing architecture, one should be prepared to re-examine and redesign existing algorithms. This should become evident as we examine various methods for computation of histograms on the GPU (a seemingly simple task that proves to be far from trivial on a massively multiprocessing architecture).

5.1 Parallel Histogram Computation

A histogram is a non-parametric density estimator which provides a consistent estimate of the pmf/pdf of the data being analyzed [81,82]. Histogram is a fundamental statistical tool for data analysis which is used as an integral part of many scientific computational algorithms. Our interest in efficient computation of histograms is for computation of the MI similarity measure which is commonly used in multi-modal image registration.

Histogram computation is straightforward on a sequential processor as shown in Listing 5.1.

```

1  for (i = 0; i < data.len; i++)
2  {
3      // 'data[]' is normalized between 0.0 and 1.0.
4      bin = data[i] * (bins - 1);
5      // 'histogram[]' is already initialized to zero.
6      histogram[bin]++;
7  }
```

Listing 5.1: A simple histogram code snippet for a sequential processor. The data is assumed to be normalized between 0 and 1.

Parallelizing a histogram with B bins over N threads is schematically shown in Fig. 5.2. The input data is distributed among threads. Each thread reads a data element from a separate stream, determines the appropriate bin associated with the data element and increments the value of the relevant histogram bin memory. Since, updates to histogram memory are data dependent, it cannot be guaranteed that threads will not attempt to increment the same memory location at the same time. If not properly handled, this results in read/write *conflicts* and will lead

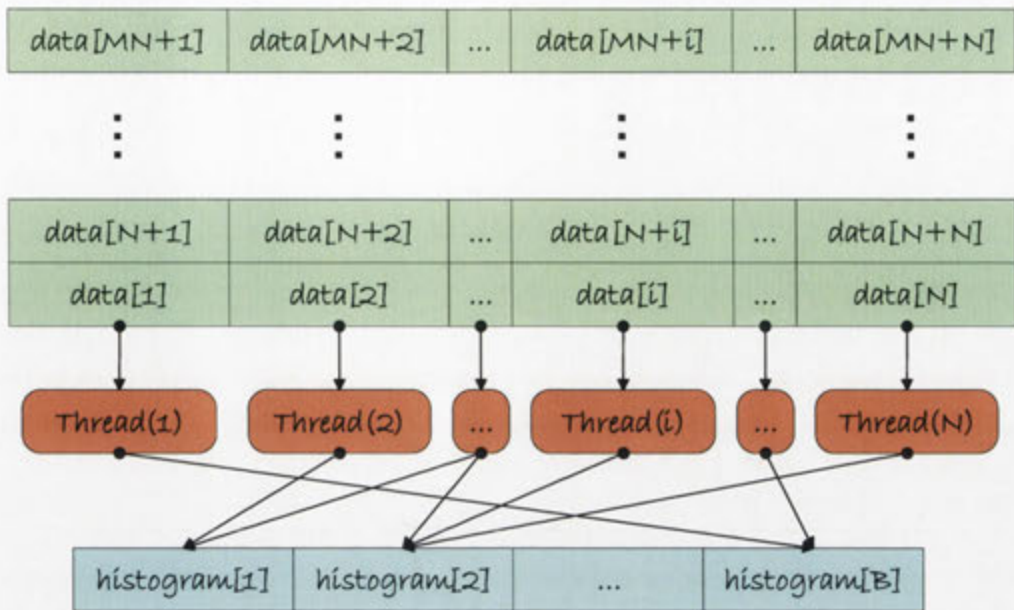


Figure 5.2: Parallel calculation of a histogram with B bins distributed to N threads, where each thread processes a vector of size $M + 1$. Histogram updates conflict and require synchronization of the threads or atomic updates to the histogram memory.

to incorrect accumulation of histogram votes. For example, consider two threads attempting to increment a histogram bin. Both threads read the current bin count c , they both increment the bin count to $c + 1$ and update the memory location to $c + 1$. Whereas the correct outcome is for the bin count to be $c + 2$.

To overcome this issue histogram updates need to be regularized across threads to ensure exclusive access by a single thread at any one time. The easiest way to achieve this is by *thread synchronization*, if supported by an architecture. Synchronization involves using appropriate primitives such as a *mutex*, a *critical section*, or *atomic operations* to serialize threads when more than one attempts to access a shared resource. The other option is to design an algorithm such that either threads do not have to share resources or the access pattern to a shared resource can be guaranteed to be conflict-free. We call this approach synchronization-free parallelization. Synchronization-free parallelization may not be always possible for an entire application. However, as we will demonstrate in the following section, it is good practice to try to minimize or remove synchronization where possible.

5.2 Parallel Histogram Computation on the CPU

Before delving into histogram implementation on the GPU, let us start by the easier task of histogram parallelization on the CPU. We compare two methods

of parallelization, with and without use of synchronization. This will allow us to demonstrate certain points regarding efficient parallelization of histograms without the clutter associated with a GPU-based method.

5.2.1 Parallelization with Atomic Operations

A naive parallelization of Listing 5.1 uses atomic operations to synchronize access to histogram memory as shown in Listing 5.2. The OpenMP directive in line 5 sets up the subsequent line as an atomic operation. This ensures that reading from histogram memory, incrementing the memory location and writing the result back is uninterrupted by other threads until completion.

```
1 #pragma omp parallel for
2 for (i = 0; i < data.len; i++)
3 {
4     bin = data[i] * (bins - 1);
5     #pragma omp atomic
6     histogram[bin]++;
7 }
```

Listing 5.2: A naive histogram parallelization with excessive use of atomic operations.

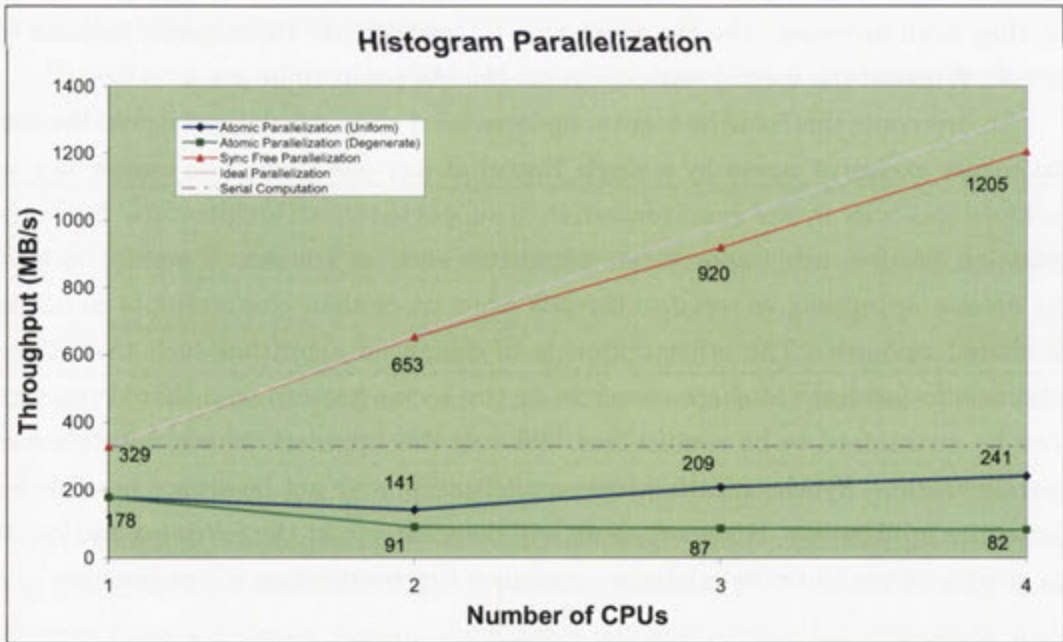


Figure 5.3: Comparison of two histogram parallelization approaches with and without synchronization. The synchronization-free approach scales almost linearly, whereas a naive parallelization by excessive use of synchronization performs worse than the baseline serial algorithm.

There are several problems with this implementation, though. Firstly, there is an overhead associated with issuing atomic operations which are being executed for every histogram update in the main loop. This slows down the computation even when there are no update conflicts and threads do not have to be serialized. In fact, the performance of Listing 5.2, for any number of CPUs, is worse than the serial implementation in Listing 5.1. We have shown the throughput of this method for 1-4 CPUs in Fig. 5.3. There is a significant performance reduction for 2 CPUs due to combined overhead of atomic operations and synchronization of threads. The performance improves slightly for 3 and 4 CPUs but would eventually fall if we could increase the number of CPUs further. As this would result in more and more threads to be serialized due to increasing rate of conflicts.

Secondly, the histogram implementation in Listing 5.2 is affected by the distribution of the data and the number of bins. In fact, the blue curve is based on a random data-set with uniform distribution which on average results in the least number of conflicts of any random distribution. The worst case scenario is given by the green line for a *degenerate* distribution where all the elements of the data-set are set to the same value. This clearly demonstrates that the method's performance is distribution dependent and is adversely affected as the number of conflicts increases. Reducing the number of bins has a similar effect as it increases the number of update conflicts.

5.2.2 Synchronization-Free Parallelization

The alternative is to devise a method that guarantees conflict-free access to histogram memory. Ideally, we would like the performance of a parallel histogram implementation to scale linearly, and be independent of the underlying data and the number of bins. We will refer to these requirements again when we discuss histogram computation methods on the GPU. A parallel implementation that meets these requirements is given in Listing 5.3. The method allocates and maintains a separate histogram per thread which is used to store a partial histogram for the portion of the data assigned to one thread. In the end, the method performs a *reduction* of partial histograms into the final histogram and requires no thread synchronization. The method's memory requirement is $\mathcal{O}(L + NB)$ as opposed to $\mathcal{O}(L + B)$ for the method in Listing 5.2, where L is the data-set length, N is the number of threads, and B is the number of bins. Effectively, only lines 20 and 21 are being parallelized and the rest can be virtually considered serial¹. The parallel

¹To be accurate the entire code is being run by N threads in parallel. However, since N partial histograms have been created, parts of the program that involve initialization and reduction of

```

1  #pragma omp parallel
2  {
3      // Allocate an array of partial histograms
4      #pragma omp single
5      {
6          N = omp_get_num_threads();
7          arrHist = new unsigned int *[N];
8      }
9
10     // Initialize each partial histogram
11     int tid = omp_get_thread_num();
12     if (tid == 0)
13         arrHist[tid] = histogram;
14     else
15         arrHist[tid] = new unsigned int[ bins ];
16     unsigned int *localHist = arrHist[tid];
17     memset(localHist, 0, sizeof(unsigned int) * bins);
18
19     #pragma omp for
20     for (int i = 0 ; i < data_len; i++)
21         localHist[(unsigned int)(data[i] * (bins - 1))]+=;
22 }
23
24 // Reduction of local histograms
25 for (int j = 1; j < N; j++)
26 {
27     unsigned int *localHist = arrHist[j];
28     #pragma omp parallel for
29     for (int i = 0; i < bins; i++)
30         histogram[i] += localHist[i];
31
32     delete [] localHist;
33 }
34 delete arrHist;

```

Listing 5.3: Histogram parallelization by reduction of partial histogram.

and serial portions of the program have a complexity of $\mathcal{O}(L/N)$ and $\mathcal{O}(B)$, respectively. As long as $L \gg NB$, efficient parallelization can be achieved. This is demonstrated in Fig. 5.3 where the synchronization-free method exhibits close to linear scalability when we increase the number of CPUs.

5.3 Histogram Computation on the GPU

Many image processing applications require histogram computation as part of a more complex task and will benefit from efficient methods for its computation on the GPU. Our motivation, however, is to allow efficient computation of mutual information (MI) for real-time registration of multi-modal images. Without an efficient histogram method on the GPU, the data needs to be moved back from the device (GPU) memory to the host (CPU), resulting in costly data transfers and reduced efficiency.

partial histograms are computationally equivalent to a single thread acting on only a single histogram.

Histograms are not a natural fit to the stream processing model for which GPUs are primarily designed. GPUs are designed to process large amounts of data efficiently where data access patterns fit within a predetermined structure and remain largely independent of each other. In histogram computation, however, the access pattern to the histogram memory cannot be determined apriori and is dependent on the value of each data element. As such, efficient computation of histograms has been traditionally difficult on the GPU [83].

The difficulty of developing efficient histogram algorithms on the GPU is evident from some of the earlier efforts to perform this task through the graphics API on the previous generation of GPUs [84,85]. Limitations with *gather* and particularly *scatter* operations (i.e. random memory reads and writes respectively) made it difficult to develop efficient histogram algorithms. In [84], Fluck et al. use a gather approach that requires $B/4$ texture fetches per data element and hence has a complexity of $\mathcal{O}(LB)$. The work by Scheuermann et al. [85] is based on a scatter approach through the use of the vertex shader and accumulates histogram counts using GPU's hardware support for color blending. Scheuermann's method has a linear complexity $\mathcal{O}(L)$ and demonstrates improved performance. Nevertheless, the performance gain is limited compared to a serial implementation (the performance is up to 3.5 better than a single core CPU of comparable technology). They also discuss a number of limitations based on the capabilities of the hardware including the accuracy of histogram computations. Fortunately, the new generation of GPUs with native support for general purpose programming offer much better flexibility and no limitations for random memory access (albeit at the expense of more costly non-coalesced memory access).

In the following sections, we present a number of methods for histogram computation on CUDA enabled GPUs. A common element of all the proposed algorithms is the need to compute several partial histograms and a subsequent reduction stage to combine them into the final result. As we showed in Section 5.2.2, this is an effective strategy in histogram parallelization and also in dealing with unnecessary synchronization as long as the total size of partial histograms remains much less than the data size. This may become an issue on the GPU where we need hundreds of blocks and thousands of threads to keep the device fully utilized. Therefore, in order to achieve the highest performance, we take the number of bins in optimizing the kernel execution configuration into account. As we will see, the choice of optimal solution for a particular problem depends on the distribution of the underlying data and the number of histogram bins required.

Another common feature of the proposed algorithms is that none depend on

availability of atomic operations and synchronization primitives with the exception of the thread join. Hence, the algorithms can be run on any CUDA-enabled device irrespective of the *compute capability* of the hardware. The first three algorithms, in particular, were designed and published in 2007 [59, 75], when the GPUs did not support atomic operations. Atomic operations in the global and shared memory have since been introduced with devices of compute capability 1.1 and 1.3, respectively. This may simplify certain aspects of code development. However, we emphasize that indiscriminate use of synchronization primitives will result in poorly parallelized programs. In the following sections, unless otherwise noted, we use GTX 8800 (see Table C.2 for hardware specifications), which does not support atomic operations, for reporting the results.

5.3.1 Method 1: Simulating Atomic Updates in Software

As mentioned before, all the threads of a warp execute the same instruction at any one time. Normally, the threads should operate on independent data elements. However, if multiple threads within a warp attempt to update the same memory location, the outcome cannot be readily predicted. We deliberately avoided labeling the outcome ‘unpredictable’ because one of the threads, and only one, will successfully update the destination [66]. So, if two threads in a warp attempt to update the same memory location at the same time with values a and b , we at least know that the memory location in question will hold either a or b after executing the instruction. We cannot predict beforehand which one will be successful but the value can be read back to determine the outcome. This behavior allows us to mimic the functionality of an atomic update as shown in Listing 5.4.

```

1  // 'bin' is defined as 'volatile' to prevent the comp-
2  //iler from optimizing away the comparison in line 12.
3  volatile unsigned int bin;
4  unsigned int tagged;
5  bin = (unsigned int) (data[i] * (bins - 1));
6  do
7  {
8      //The lower 5 bits of the thread id (tid) are
9      //used to tag the memory location.
10     tagged = (tid << 27) + (histogram[bin] + 1);
11     histogram[bin] = tagged;
12 } while (histogram[bin] != tagged);

```

Listing 5.4: Simulating atomic updates to the shared memory for threads that belong to the same warp.

In line 5 of the above listing, a histogram bin is determined from input data. The corresponding memory location in the ‘histogram[]’ array is then *atomically*

incremented by one in lines 6-12. Each thread in a warp with w threads can be represented by a $\lceil \log_2 w \rceil$ -bit identifier where $\lceil \cdot \rceil$ is the ceiling operator. When a thread attempts to write into the histogram it also tags the memory location by combining the histogram value with its identifier into a 32-bit word². The code assumes that there are 32 threads in a warp and hence, the identifier and the histogram values are allocated 5, and 27 bits, respectively. In line 10, the current value of the histogram bin is incremented by all participating threads in the warp and tagged with the lower 5 bits of the thread ID. The result is then stored in a variable named 'tagged'. The result is written back into the histogram array in line 11. This is where the update conflicts may occur if the value of 'bin' is the same for more than one thread. In case of an update conflict, one of the threads will succeed in updating the histogram and will leave its thread ID behind. In line 12 each thread will determine whether it has successfully updated the histogram or not. Threads that have been successful will no longer execute the update loop and will allow the remaining threads to try their chance in the next iteration.

The update loop will be executed up to c times, where c is the maximum number of threads that are attempting to update a single bin. The upper bound for c equals the number of threads in a warp. If supported by the hardware, the entire update loop can be replaced by a hardware-based atomic addition. Either way, the efficiency of the algorithm is dependent on the number of update conflicts which in turn is dependent on the distribution of the underlying data and the number of bins. The worst case scenario occurs when all data elements belong to the same bin (e.g. when data has a *degenerate* distribution i.e. all the data elements are the same).

For this method to work one needs to ensure that the 'histogram[]' array is only updated by one warp at a time. Hence, we need to maintain separate partial histograms per warp. Partial histograms have to be allocated on the GPU's shared memory for this method to be efficient. However, given the 16 KB limit for the shared memory, the maximum number of 32-bit bins that can be supported is 4096³. Allocating 'histogram[]' in the global memory to overcome this limit is not advisable as repeated updates to the high latency global memory will significantly reduce the execution speed.

We also note that using a single warp under-utilizes the GPU resources. For optimal performance the GPU needs around 4-8 warps. As such, we need to allocate

²As a result of tagging the histogram, the frequency of data samples may not exceed $2^{32 - \lceil \log_2 w \rceil}$.

³The actual number is slightly lower, since CUDA uses shared memory to pass arguments and execution information to the kernel.

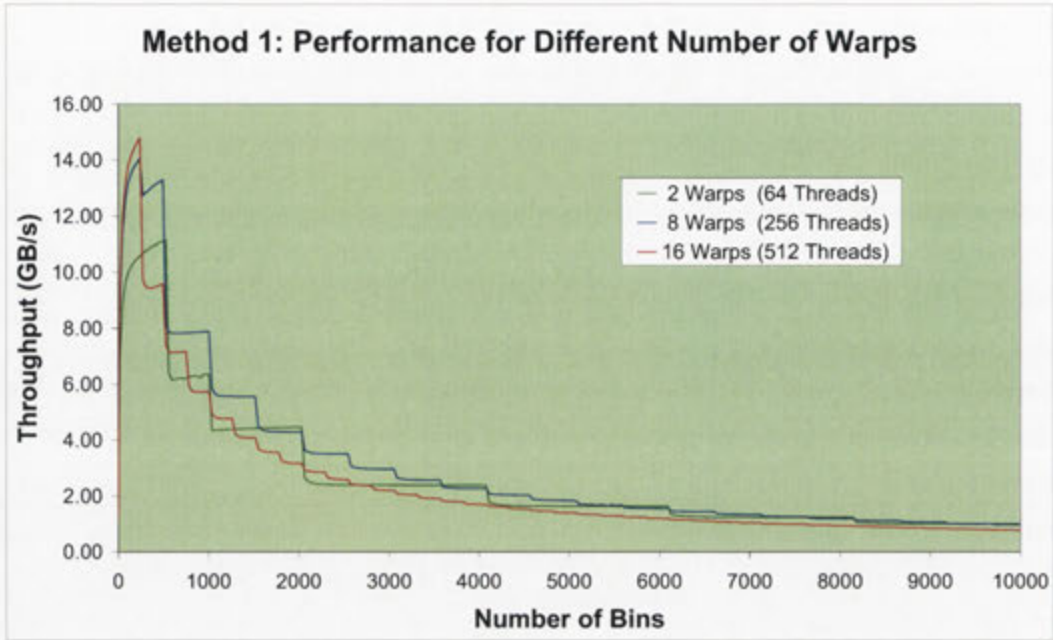


Figure 5.4: The maximum performance of the method is higher with more warps but drops more quickly with the number of bins. Lower number of warps perform better for larger bins.

a separate histogram array for each warp. The sub-histogram arrays are then combined to produce the final result. Obviously, increasing the number of warps will further limit the number of bins that can be processed per execution of the algorithm.

To allow histogram computation with an arbitrary number of bins, we divide the bin ranges into a number of sub-ranges that fit in the shared memory. For a given execution configuration, the algorithm is then run as many times as required to cover the entire bin range. At each iteration the kernel will only process those data elements which fall in the specified bin range. For example, with 4 warps and a limit of 1024 bins per execution, a 10,000 bin histogram requires 10 iterations of the algorithm.

Fig. 5.4 shows histogram computation throughput in gigabytes per second for a data-set with a uniform distribution and for different number of warps. The number of bins is varied from 1 to 10,000. Higher number of warps results in improved performance per iteration of the algorithm. This can be seen in Fig. 5.4 for smaller bins, where the computation can be completed in a single iteration. The sharp drops in throughput in each curve correspond to points where the total number of bins for a given number of warps exceeds the shared memory size and the algorithm has to perform an additional iteration to complete. Hence, as the

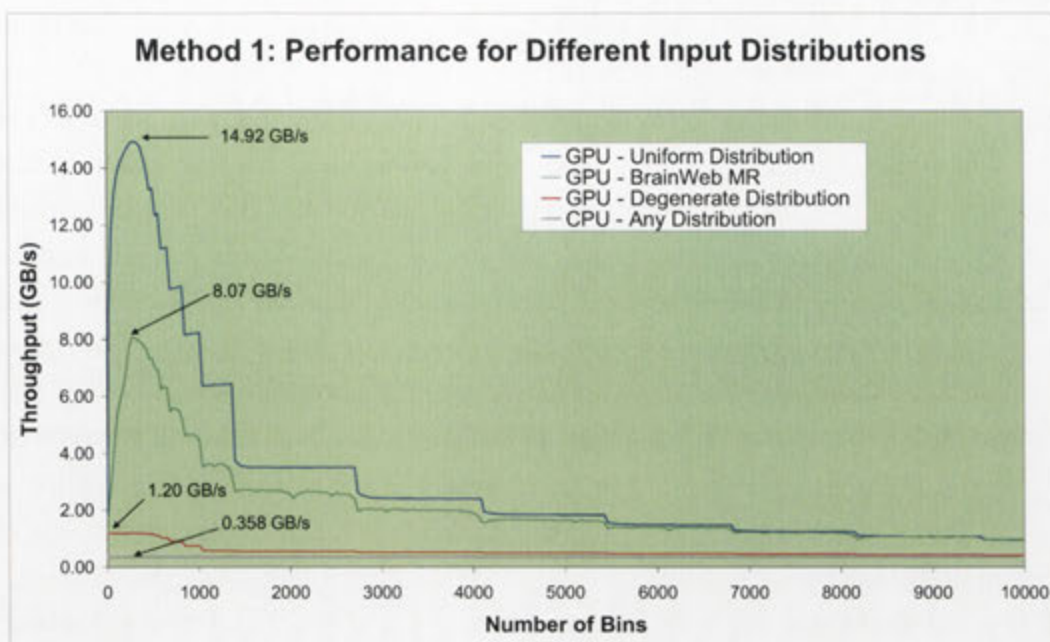


Figure 5.5: The performance of method 1 is dependent on the distribution of the input data. Data with a uniform distribution performs best. The worst performance is observed for a degenerate distribution. The performance for practical applications is somewhere between these upper and lower bounds. The GPU gives up to 42 times improved performance compared to a single core CPU

number of bins increases, the number of warps has to be reduced to delay reaching such a breakpoint at the expense of under-utilizing the GPU.

To achieve an optimal throughput, we choose the number of warps w_{opt} depending on the number of bins such that

$$w_{\text{opt}} = \underset{w}{\operatorname{argmax}} wB, \quad wB \leq s_{\text{max}}, \quad 3 \leq w \leq w_{\text{max}}, \quad (5.2)$$

where w is the number of warps, B is the number of bins, s_{max} is the maximum number of double words that can be allocated in the shared memory, and w_{max} is the maximum number of warps supported by the hardware. We also found that the best utilization of the GPU can be achieved by setting the number of blocks to three times the number of available multiprocessors.

As can be seen in Fig. 5.4, the performance of the method decreases with increasing number of bins with the exception of the very beginning of the bin range where the performance decreases as we reduce the number of bins. This anomaly is due to an increased rate of update conflicts when the data elements are being mapped to fewer bins.

In Fig. 5.5, we have shown the throughput of the method for a random input with a uniform distribution, a sample MR image, and a degenerate distribution. The random input with uniform distribution is close to the best case scenario⁴, as for large inputs the histogram is going to be uniform and the histogram update collisions are close to minimal. A degenerate distribution results in maximum histogram update collisions, as all the threads try to update the same histogram bin and as such represents the worst case scenario. The performance for a real application is somewhere in between these lower and upper bounds. We have represented this using data from an MR image. For comparison, we have also shown the throughput of histogram calculation on a single core CPU (refer to Appendix C for specifications). The performance of the histogram on the CPU is almost constant w.r.t. the number of bins and is not affected by the distribution of the input data.

Fig. 5.5 shows that the GPU implementation of method 1 has a clear advantage, especially for smaller bins. However, for a sufficiently large number of bins the method will inevitably lose its performance benefit over a single core CPU. Another, disadvantage is that as with any other atomic-based method, the performance is dependent on the underlying data-distribution. We address the latter problem in the next section.

5.3.2 Method 2: Synchronization-Free Parallelization

The second method that we discuss is functionally equivalent to the synchronization-free parallelization method on the CPU given in Section 5.2.2. Similar to the CPU version, a partial histogram is maintained for every thread. The partial histograms are combined at the end to obtain the final result. The partial histograms have to be created in the global memory because the shared memory is simply not large enough to hold a partial histogram for every thread for a reasonably sized histogram.

The benefit is that, given the size of the global memory, for any practical number of bins the algorithm only requires a single iteration to complete. In addition, there will be no concurrent updates of the same memory location by multiple threads and as such no update synchronization is required, which in turn means that the performance of the method is not data dependent⁵.

However, there are two drawbacks to this method; firstly, a much larger memory

⁴The best performance is achieved by a data-set ‘engineered’ to map into different bins for every chunk of data processed by a warp.

⁵To be more precise, the method is not completely data-independent. Bin-packing (explained later) introduces some dependence on the data distribution.

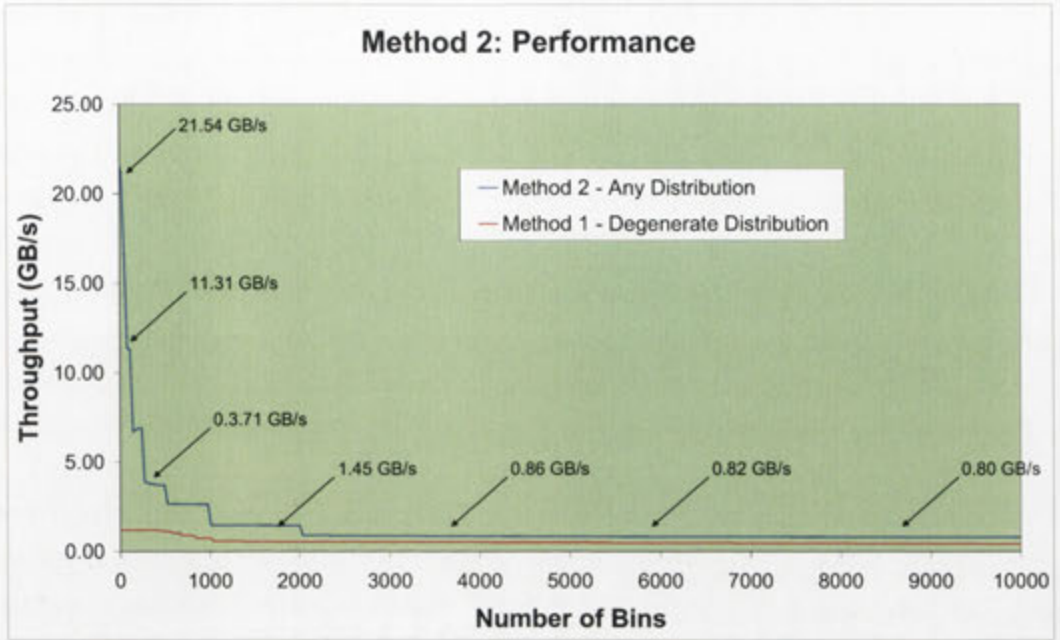


Figure 5.6: The performance of method 2 is above the first method's worst case and the CPU-based histogram implementation. The maximum throughput is achieved for very small bins with an improvement of up to 60 times compared to the CPU implementation.

for partial histograms needs to be allocated and initialized to zero at the beginning; secondly, histogram updates need to be done on the global memory, this entails non-coalesced read/writes per input element and is inefficient.

The standard memory initialization method, '*cudaMemset*', proved to be inefficient. We implemented a method for initializing an array in the kernel with a throughput of around 60 GB/s, which to some extent addressed the first problem.

To address the second problem and avoid excessive non-coalesced updates of the global memory, a *delayed-write* technique is employed. We pack multiple bins in a 32-word in the shared memory and only update the corresponding bin in the global memory when the packed bin overflows. This reduces the updates to the global memory by a factor of 2^i , where i is the number of bits available for storage of a bin in the shared memory. The number of available bits depends on the number of threads per block and the number of bins and is calculated as

$$i = \left\lfloor \frac{s_{\max} \times 32}{B \times N_b} \right\rfloor, \quad (5.3)$$

where s_{\max} is the maximum number of double words that can be allocated in the shared memory, B is the number of bins and N_b is the number of threads per block.

There is a trade-off between the number of threads and the number of bits per bin. Increasing the number of threads, decreases i , resulting in more global memory updates, while reducing the number of threads can under-utilize GPU resources and affect the performance. We optimized these parameters for the best performance, the result is shown in Fig. 5.6. As can be seen method 2 performs consistently better than the method 1's worst case. The performance of method 2 for smaller bins is better or comparable with method 1's best case. However, this method under-performs method 1's best case for the mid and high bin ranges.

5.3.3 Method 3: Approximate Histogram

The third method that we discuss is not a histogram computation method per se, rather it obtains an approximation of the input's distribution function. As we indicated earlier, our primary application for histogram computation is to obtain an estimate of the marginal and joint pmf of multi-modal images for MI-based registration. Hence, it is not necessary to limit ourselves to an exact computation of histograms as long as we can reasonably estimate the probability distribution of the images.

Starting with the assumption that a histogram is a reasonable approximation of the pmf of a discrete random signal, one would expect to estimate a more or less similar pmf from a sufficiently large subset of observations. As long as the observations are independent, it should not matter what subset of the signal is being used for the estimation. We use this intuition to our advantage in order to improve the performance of pmf computation on the GPU by designing a sampling strategy that guarantees conflict-free updates of partial histograms by the threads of a block.

The pmf calculation is to be distributed to K thread blocks each with N_b threads. Each block will maintain a partial histogram of its own in the global memory for the portion of input data assigned to the block. The entire bin range is divided into N_b non-overlapping regions. Each thread in a block is responsible for computing the histogram count for one of these non-overlapping regions. The bin range assigned to each thread ID t_{id} (zero-indexed) is given by

$$\lfloor \frac{B}{N_b} \times t_{id} \rfloor \leq b < \lfloor \frac{B}{N_b} \times (t_{id} + 1) \rfloor, \quad (5.4)$$

where, B is the number of bins, N_b is number of threads per block, $\lfloor a \rfloor$ denotes the largest integer value that does not exceed a . Based on (5.4), each thread will handle either $\lfloor \frac{B}{N_b} \rfloor$ or $\lfloor \frac{B}{N_b} \rfloor + 1$ bins.

Fig. 5.7 shows data access and execution configuration for each block. Input data allocated to each block is further divided among the threads. Each thread will process data only if the histogram bin b for a data element falls within the subset of bins assigned to that particular thread. Therefore, a single partial histogram per block is required. The algorithm guarantees that histogram updates by different threads are non-coincident. We also note that the method samples the input data at a rate approximately equal to $\frac{1}{N_b}$.

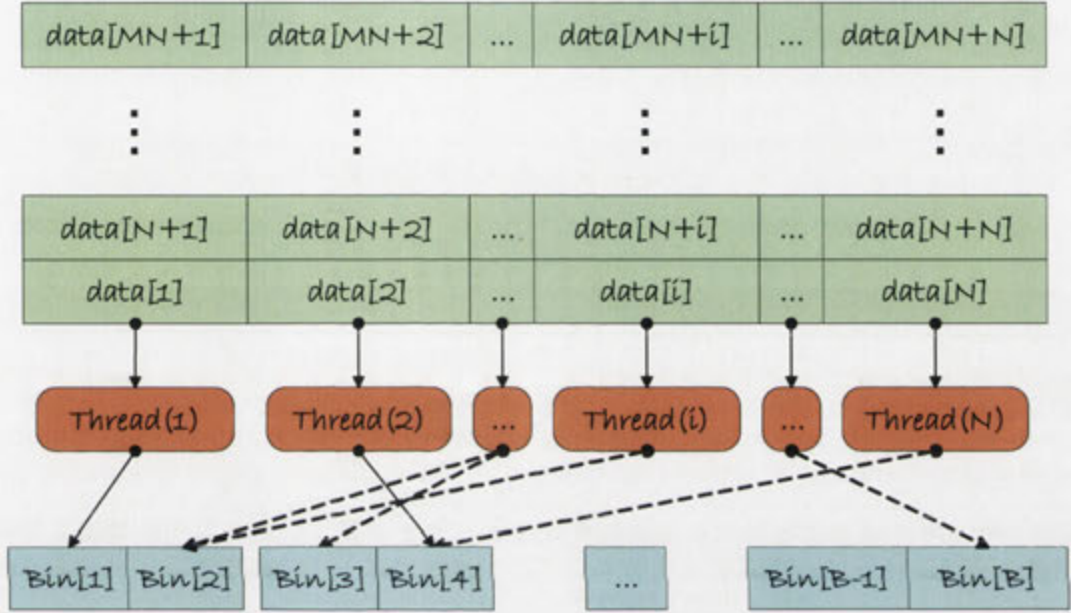


Figure 5.7: Data access and execution configuration for threads within a block in method 3. Schematically, we have displayed allocation of two bins per thread. Dotted lines indicate that the data is outside the bin range assigned to the thread and as a result is discarded.

To further improve the performance, we adopt the delayed-write technique used in method 2. Depending on the number of bins several bins may be packed in a single 32-bit word to hold a partial count of the histogram. Whenever, a packed bin overflows, we increment the corresponding partial histogram in the global memory. This method delays global memory updates to the extent possible. For example, for 10,000 bins we can allocate 8-bits per bin in the shared memory. This means that the block's partial histogram has to be updated once 256 samples are accumulated in a given bin.

Typically, 128-256 threads per block are required to ensure that the GPU's computing resources are fully utilized. Fig. 5.8 depicts the throughput of method 3 for different number of threads and shows that the method is scalable and the performance increases as we increase the load on the GPU. However, increasing the

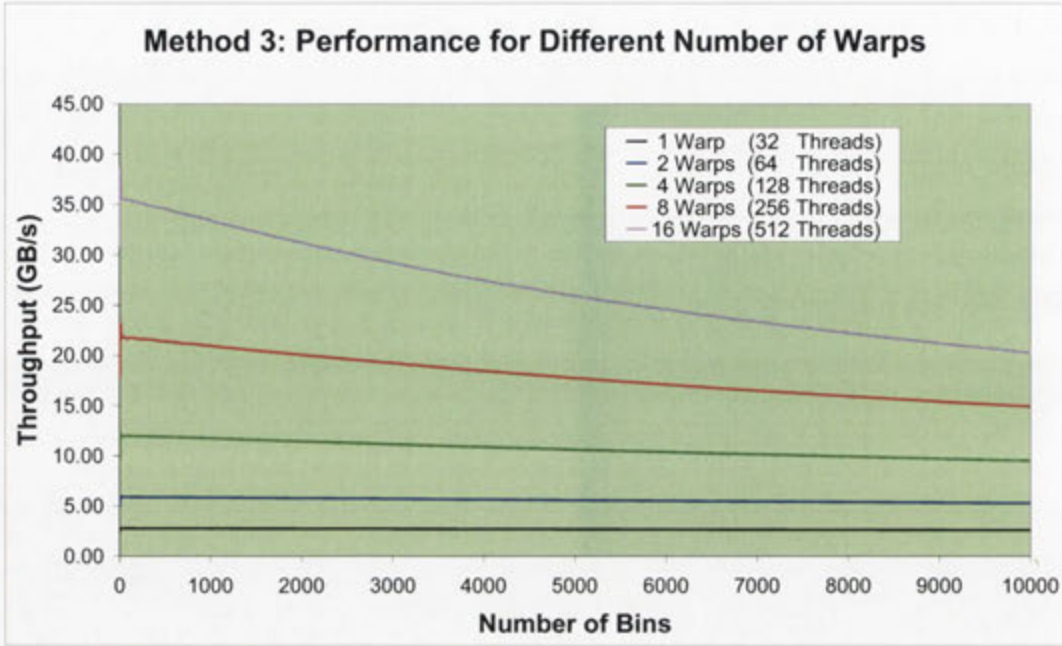


Figure 5.8: Method 3 scales well as the number of warps is increased.

number of threads results in a lower sampling rate and may result in a less accurate estimation of the histogram. A right balance between the required accuracy and the computational efficiency must be struck. As a rule of thumb, the larger the data-set and the less dependent the observations, the lower the sampling rate can be set.

The main benefit of this approximate method is that it maintains its performance for a wide range of histogram bins. The performance is also virtually unaffected by the distribution of the underlying data stream.

5.3.4 Method 4: Sort and Count

The final method that we discuss maintains the desirable properties of the approximate histogram algorithm (i.e. its performance is largely unaffected by the number of bins and distribution of data) with the added benefit that it computes an exact histogram. The main idea here is to reorganize input data in a way that facilitates histogram computation and removes the need for atomic operations or synchronization at the same time. The method is based on the *sort and count* algorithm that we first presented in [63].

The previous methods that we discussed were specifically designed for CUDA-enabled GPUs. One can argue that hardware-specific algorithms will inevitably be superseded due to improved capabilities of future generations of hardware. The

argument was certainly stronger for previous generations of GPUs with limited general purpose programming capabilities. This is less of an issue for modern GPUs where the programming environment is C/C-like and there is an upgrade path to future generations of hardware. Nevertheless, there is more intrinsic value in devising algorithms designed for massively parallel architectures with no dependence on the specifics of a particular hardware. This is the approach taken with sort and count method, where the proposed algorithm is general and hardware-independent and can be applied to an arbitrary platform. For comparison with previous methods, we still develop and test the algorithm on NVIDIA hardware. However, we start by introducing the concept of the algorithm first, and leave the details of a CUDA specific implementation to later subsections.

Sort and Count Algorithm

The purpose of a sort and count algorithm is to determine the number of equally valued elements of a data-set while the data-set is being sorted by a suitable algorithm. We begin by defining a number of concepts.

Let $I = \{1, 2, \dots, n\}$ be an index set and A be a set with the same cardinality as I ($|A| = n$) and with a *total order*⁶ defined in terms of a comparison operator \leq . A sequence s is defined as a one-to-one mapping $s : I \rightarrow A$. A sort algorithm arranges an arbitrary sequence

$$s : \{a_1, a_2, \dots, a_n\}, \quad a_i \in A$$

to a new sequence

$$s' : \{a_{\phi(1)}, a_{\phi(2)}, \dots, a_{\phi(n)}\}$$

such that

$$a_{\phi(i)} \leq a_{\phi(j)}, \quad \text{for all } i < j,$$

where ϕ is a permutation of the index set I .

A *binary comparison-only* sort algorithm, is one that reorganizes a sequence using knowledge gained solely by application of a comparison operator on pairs of records. A sort algorithm is *stable* if it maintains the relative order of elements with equal keys (e.g. values) [86]. An arbitrary sort algorithm with a given comparison

⁶A total order is a binary relation that is transitive ($x \leq y, y \leq z \rightarrow x \leq z$), antisymmetric ($x \leq y, y \leq x \rightarrow x = y$), and total (either $x \leq y$ or $y \leq x$). A *partial order*, such as the subset operator (\subset), does not possess the last property.

operator \leq can be *stabilized* by introducing a new comparison operator \prec such that

$$a_i \prec a_j \iff (a_i < a_j) \vee (a_i = a_j \wedge i < j). \quad (5.5)$$

Note that no two keys are equal under the new comparison operator.

Consider a stabilized binary comparison-only sort algorithm such as S . We introduce a modification to S so that it counts the number of elements that are equal under the original comparison operator while sorting. We assign a counter to each element and initialize the counters to 1 at the beginning

$$s : \{a_1^{(1)}, a_2^{(1)}, \dots, a_n^{(1)}\},$$

where s is the initial sequence and the superscripts denote the counters. The counters are updated every time a comparison is performed if $a_i^{(n_i)} = a_j^{(n_j)}$ with $i < j$ such that

$$\begin{aligned} n_j &\leftarrow n_j + n_i, \\ n_i &\leftarrow 0. \end{aligned} \quad (5.6)$$

In other words, every time a comparison of equal elements is performed, the element with a higher position in the sequence accumulates the counter of the lower element. We call this a *sort and count* algorithm.

For example, performing a sort and count on a sequence such as $\{1, 3, 1, 1, 2, 3\}$ results in the following sequence: $\{1^{(0)}, 1^{(0)}, 1^{(3)}, 2^{(1)}, 3^{(0)}, 3^{(2)}\}$. The sort and count steps for this example are given in Table 5.1 using a simple *bubble sort* algorithm (in practice, we will use a parallel sort algorithm but it is easier to demonstrate the concept using a simpler sort algorithm such as the bubble sort). The sequence is sorted after 8 iterations, however bubble sort will take another iteration to realize this. As can be seen in row 8 of Table 5.1, at the end of the sort and count algorithm, numbers with the highest position in their groups have their counters set to the number of group elements.

Theorem: At the completion of a sort and count algorithm, all elements within a subset with equal values have their counters set to zero except for the element with the highest position in the sequence whose counter contains the cardinality of the subset.

Proof: Let us denote an arbitrary subset of equally valued elements (if one exists) within the sorted sequence by $\{b_{k+1}^{(n_{k+1})}, b_{k+2}^{(n_{k+2})}, \dots, b_{k+m}^{(n_{k+m})}\}$. For an element

Table 5.1: An example of a bubble sort and count for a simple sequence

Itr.	Sequence	Compare	Result
1	$\{\underline{1}^{(1)}, \underline{3}^{(1)}, 1^{(1)}, 1^{(1)}, 2^{(1)}, 3^{(1)}\}$	(1, 3)	$1 < 3$, no order change
2	$\{1^{(1)}, \underline{3}^{(1)}, \underline{1}^{(1)}, 1^{(1)}, 2^{(1)}, 3^{(1)}\}$	(3, 1)	$3 > 1$, swap
3	$\{1^{(1)}, 1^{(1)}, \underline{3}^{(1)}, \underline{1}^{(1)}, 2^{(1)}, 3^{(1)}\}$	(3, 1)	$3 > 1$, swap
4	$\{1^{(1)}, 1^{(1)}, 1^{(1)}, \underline{3}^{(1)}, \underline{2}^{(1)}, 3^{(1)}\}$	(3, 2)	$3 > 2$, swap
5	$\{1^{(1)}, 1^{(1)}, 1^{(1)}, 2^{(1)}, \underline{3}^{(1)}, \underline{3}^{(1)}\}$	(3, 3)	$3 = 3$, no order change, higher index 3 accumulates the counter
6	$\{\underline{1}^{(1)}, \underline{1}^{(1)}, 1^{(1)}, 2^{(1)}, 3^{(0)}, 3^{(2)}\}$	(1, 1)	$1 = 1$, no order change, higher index 1 accumulates the counter
7	$\{1^{(0)}, \underline{1}^{(2)}, \underline{1}^{(1)}, 2^{(1)}, 3^{(0)}, 3^{(2)}\}$	(1, 1)	$1 = 1$, no order change, higher index 1 accumulates the counter
8	$\{1^{(0)}, 1^{(0)}, \underline{1}^{(3)}, \underline{2}^{(1)}, 3^{(0)}, 3^{(2)}\}$	(1, 2)	$1 < 2$, no order change
9	$\{1^{(0)}, 1^{(0)}, 1^{(3)}, \underline{2}^{(1)}, \underline{3}^{(0)}, 3^{(2)}\}$	(2, 3)	$2 < 3$, no order change

such as $b_i^{(n_i)}$ with $i \neq k + m$ to have a non-zero count, it must have never been compared with $b_{i+1}^{(n_{i+1})}, \dots, b_{k+m}^{(n_{k+m})}$, otherwise the count would have been accumulated by the higher position element. Now let us update b_i in the original sequence with $b_i + \epsilon$ such that $\epsilon > 0$ and $b_i + \epsilon < b_{k+m+1}$. If we run the stabilized sort algorithm again, this change will not affect the outcome, because the updated element is still less than b_{k+m+1}, \dots, b_n , it is greater than b_{i-1}, \dots, b_1 and since it is never compared against b_{i+1}, \dots, b_{k+m} , the algorithm makes the exact same decisions at each step and hence b_i must occupy the exact same position. However, this also means that the algorithm has failed to sort the sequence ($b_i + \epsilon > b_{i+1}$) and the proof is complete.

Sort and Count for Parallel Histogram Computation

For an arbitrary data-set, one can perform the sort and count algorithm on the data-set using the bin values corresponding to each element as the sort key. At the completion of this process, the non-zero counters associated with the sorted sequence contain the value of corresponding histogram bins.

With a sort algorithm that can be efficiently parallelized (such as the bitonic sort described later), sort and count can be used to efficiently parallelize histogram

computation for massively multiprocessing architectures. Since the outcome of sort contains a single non-zero counter for each unique histogram bin, a number of threads can process the non-zero counters in parallel and update histogram locations corresponding to these bin values. The algorithm is guaranteed to be free of update conflicts and has no reliance on synchronization primitives. Of course, in practice, we do not sort the entire input data, the data is read in blocks, sorted and the histogram is updated with the counters in each iteration. Fig. 5.9 demonstrates the concept of updating the histogram for the simple example of the previous section, where each thread reads a bin from the sorted sequence and only writes to the histogram memory if the bin contains a non-zero count. As can be seen, some threads may not perform an update and remain idle, however this is a big improvement over having to serialize them.

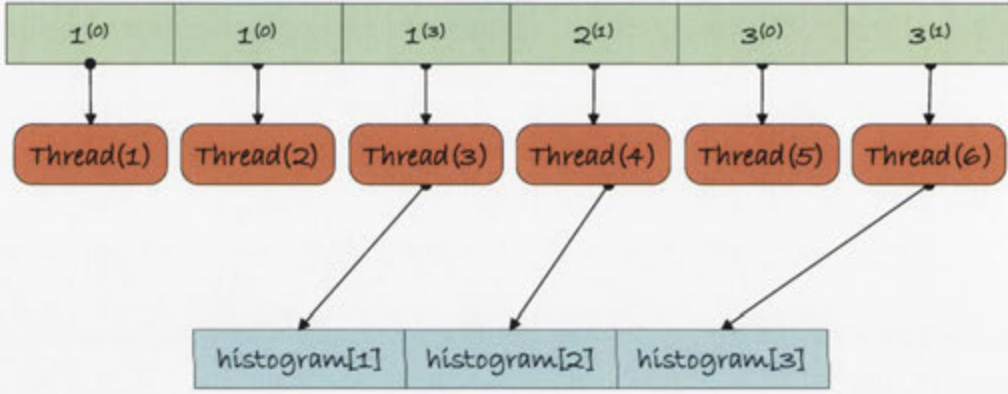


Figure 5.9: Conflict-free update of the histogram with sort and count.

Bitonic Sort Algorithm

We used ‘bitonic sort and count’ to test the performance of the proposed method. In this section, we provide a brief overview of the bitonic sort algorithm itself [87].

A sequence of non-decreasing or non-increasing numbers is called *monotonic*. A *bitonic* sequence is one that consists of an ascending and a descending monotonic sequence. A *bitonic sorter* converts a bitonic sequence into a monotonic sequence. If a bitonic sequence of $2n$ numbers, $\{a_1, a_2, \dots, a_{2n}\}$, is reorganized into

$$\{\min(a_1, a_{n+1}), \min(a_2, a_{n+2}), \dots, \min(a_n, a_{2n}), \\ \max(a_1, a_{n+1}), \max(a_2, a_{n+2}), \dots, \max(a_n, a_{2n})\},$$

the new sequence is also bitonic and none of the elements in the first half of the sequence can be greater than any elements in the second half [87]. This is known

as the *bitonic merge* theorem. So given a bitonic sequence of size $2n$ one can sort the sequence by merging the sequence first and using 2 bitonic sorters of size n . This provides an iterative algorithm for sorting a sequence of $L = 2^m$ elements using a bitonic sorter. Fig. 5.10 shows a bitonic sort network for an input sequence of 8 elements which completes after 6 iterations regardless of the input sequence. The order of comparisons at each iteration is pre-determined as shown in Fig. 5.10. Bitonic sort is not an optimal sort⁷ and requires $\mathcal{O}(L(\log L)^2)$ comparisons. However, since the sequence of comparisons is predetermined and data-independent, the algorithm can be efficiently parallelized.

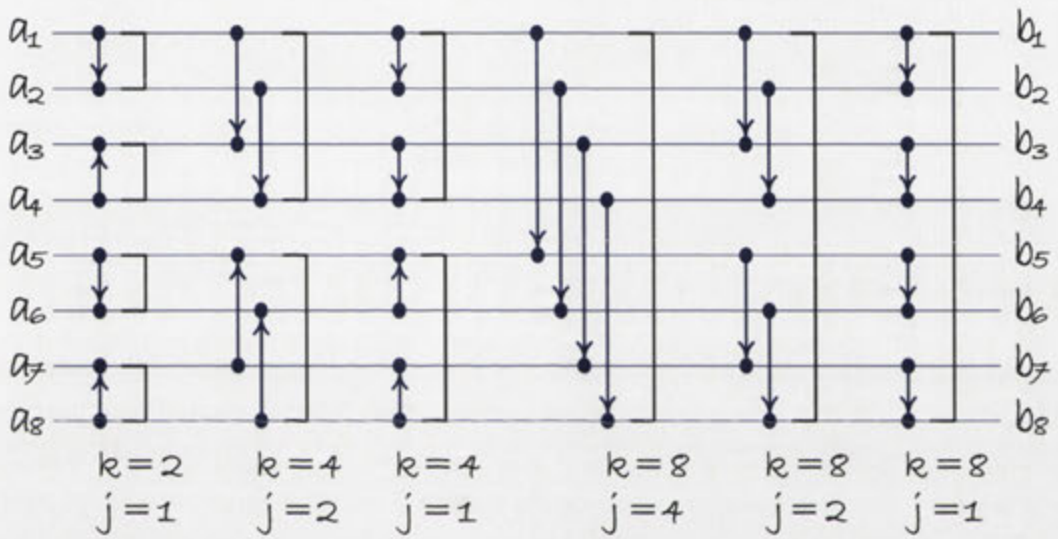


Figure 5.10: Ascending bitonic sort for a sequence of 8 elements. Each connector represents a comparison in the direction denoted by the arrow. Elements involved in each comparison are swapped if they are not in the desired order. k and j represent block size and comparison offset at each level, respectively.

CUDA Implementation

In our CUDA implementation, the data is distributed among k blocks each with N_b threads which process input data in chunks of $2N_b$ size in each iteration. Each block maintains a partial histogram in the global memory. The partial histograms are combined using a parallel *reduction* algorithm at the end. Input values are stored as 32-bit unsigned integer values, and counters are packed into the higher bits of the same memory location, for most efficient handling.

Listing 5.5 shows the kernel function executed by each thread to perform a bitonic sort and count algorithm. Bins and their associated counters are packed into

⁷An optimal sort algorithm executes in $\mathcal{O}(L \log L)$ time.

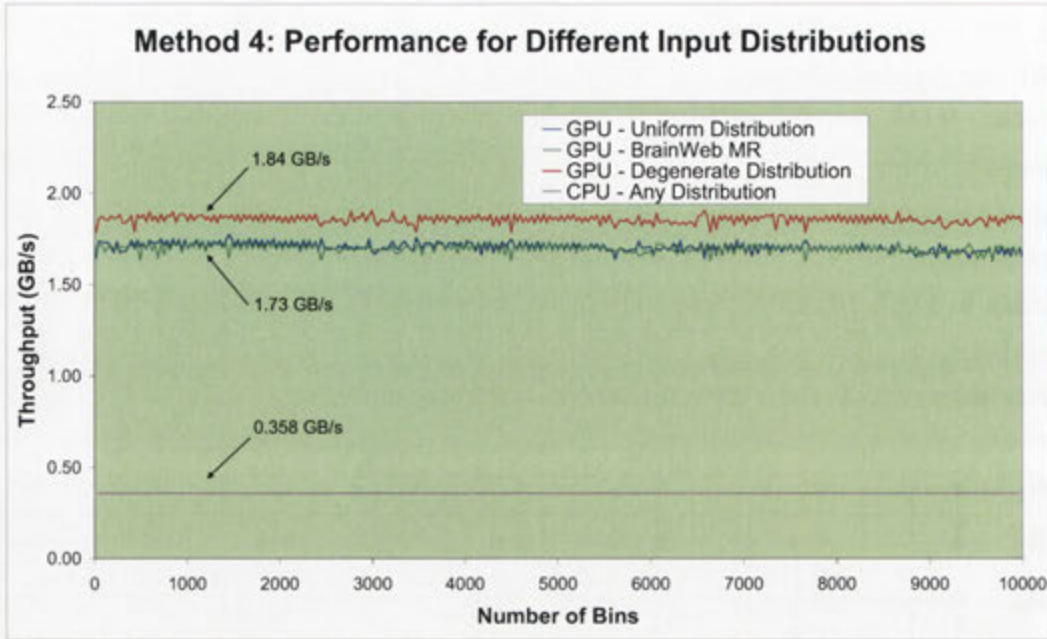


Figure 5.11: Sort and count exhibits a consistent performance for the entire bin range and is largely unaffected by the distribution of the input data.

32-bit unsigned integer variables, where the lower 16-bit word is used to store the bin number and the higher 16-bit word contains the counter value. The complete source code can be found online at <http://cecs.anu.edu.au/~ramtin/cuda/>.

We have shown the performance of the CUDA implementation on a GTX 8800 in Fig. 5.11. The performance of the method is the same for a uniform distribution and a sample MR image. The performance is slightly higher for a degenerate data-set which actually represents the method's upper-bound throughput. This is due to the fact that for a degenerate data-set, the underlying sort method does not have to swap the sequence elements which results in a slightly improved performance.

Counting a Sorted Sequence in Parallel

The sort and count method limits the class of sort algorithms that can be used as discussed in Section 5.3.4. This excludes some popular parallel sort methods such as the radix sort. Here, we present an algorithm that counts an already sorted sequence in linear time. This allows an arbitrary sort algorithm to be used for histogram computation. However, we note that the sort and count algorithm is the preferred method for counting a sequence where the sort algorithm meets the requirements of Section 5.3.4.

Consider a sorted sequence $s : \{a_1^{(n_1)}, a_2^{(n_2)}, \dots, a_n^{(n_n)}\}$ where each element has been assigned a counter as before. We call the sequence *counted* if for an arbitrary

```

1 __device__ inline void bitonicSort
2 (
3     uint *shared,           // Pointer to data in the shared memory
4     const uint tid,         // ID of the current thread
5     const uint threads      // Number of threads in each block
6 )
7 {
8     // Parallel bitonic sort
9     // k is the bitonic block length
10    // Save log2(k) to speed up the computations
11    uint log2k = 1;
12    for (uint k = 2; k <= threads << 1; k <= 1, log2k++)
13    {
14        // Bitonic merge
15        // Bitonic block Id
16        uint b_id = tid >> (log2k - 1);
17        // j is the bitonic offset
18        // Save log2(j) to speed up the computations
19        uint log2j = log2k - 1;
20        for (uint j = k >> 1; j > 0; j >= 1, log2j--)
21        {
22            uint i1 = ((tid >> log2j) << (log2j + 1))
23                + (tid & (j - 1));
24            uint i2 = i1 + j;
25            uint L1 = shared[i1] & 0x0000ffff;
26            uint L2 = shared[i2] & 0x0000ffff;
27            // Even blocks are sorted in descending order
28            if ((b_id & 1) == 0)
29            {
30                if (L1 > L2)
31                    swap(shared[i1], shared[i2]);
32                else if (L1 == L2)
33                {
34                    // Accumulate counters in i2
35                    shared[i2] = (shared[i1] & 0xffff0000)
36                        + (shared[i2] & 0xffff0000) + L1;
37                    shared[i1] = L1; // Reset i1's counter
38                }
39            }
40            // Odd blocks are sorted in ascending order
41            else
42            {
43                if (L1 < L2)
44                    swap(shared[i1], shared[i2]);
45                else if (L1 == L2)
46                {
47                    // Accumulate counters in i1
48                    shared[i1] = (shared[i1] & 0xffff0000)
49                        + (shared[i2] & 0xffff0000) + L1;
50                    shared[i2] = L1; // Reset i2's counter
51                }
52            }
53            // All threads join here
54            __syncthreads();
55        }
56    }
57 }

```

Listing 5.5: Parallel bitonic sort and count.

subsequence of equally valued elements $s_k : \{a_k^{(n_k)}, a_{k+1}^{(n_{k+1})}, \dots, a_{k+m-1}^{(n_{k+m-1})}\}$ there exists one and only one nonzero counter $k \leq j < m$ whose value equals the number of elements in the subset $(n_j = m, n_i = 0, k \leq i < m, i \neq j)$. We say a subsequence is counted to the left if $j = k$, or in other words the sequence's length is accumulated by the leftmost element. Similarly the sequence is said to be counted to the right if $j = m - 1$. We say a sequence is *bitonically* counted if the leftmost subsequence of identical values in the sequence is counted to the left and the rightmost sequence is counted to right. By convention, a sequence which is entirely composed of identically valued elements is counted to the left.

Two bitonically counted sequences $s_1 : \{a_1^{n_1}, \dots, a_\ell^{n_\ell}\}$ and $s_2 : \{a_{\ell+1}^{n_{\ell+1}}, \dots, a_n^{n_n}\}$ where $a_\ell \leq a_{\ell+1}$ can be merged in to a new bitonically counted sequence by the following updates to the sequence counters:

1. if $a_\ell = a_{\ell+1}, a_\ell = a_1$ then $n_1 \leftarrow n_1 + n_\ell + n_{\ell+1}, n_\ell \leftarrow 0, n_{\ell+1} \leftarrow 0$.
2. if $a_\ell = a_{\ell+1}, a_\ell \neq a_1, a_\ell = a_n$ then $n_n \leftarrow n_n + n_\ell + n_{\ell+1}, n_\ell \leftarrow 0, n_{\ell+1} \leftarrow 0$.
3. if $a_\ell = a_{\ell+1}, a_\ell \neq a_1, a_\ell \neq a_n$ then $n_\ell \leftarrow n_\ell + n_{\ell+1}, n_{\ell+1} \leftarrow 0$.
4. if $a_\ell \neq a_{\ell+1}, a_{\ell+1} = a_n : n_n \leftarrow n_n + n_{\ell+1}, n_{\ell+1} \leftarrow 0$.

The method can be used to recursively count an arbitrary sequence in parallel starting with subsequences of length two and merging until the entire sequence is counted. Counting a sequence of length 2^p can be efficiently parallelized by up to 2^{p-1} threads where the first iteration requires 2^{p-1} sequence merges, the second iterations requires 2^{p-2} sequence merges, etc. The algorithm completes in $p - 1$ iterations and has a linear time complexity.

5.3.5 Comparison of Histogram Computation Methods

The properties of different histogram computation methods that we have discussed in this chapter have been summarized in Table 5.2 for quick reference. The choice of the most suitable method for a particular application depends on several factors including the number of bins required, type of data, and whether an exact histogram computation is desired. The information in Table 5.2 together with throughput graphs of earlier sections should help with choosing the right algorithm for a particular problem.

Table 5.2: Comparison of histogram computation methods on the GPU

Method	Pros	Cons
1 - Simulating atomic updates in software	<ul style="list-style-type: none"> • Exact computation of histograms • Fast for low and mid bin ranges for uniform and normal distributions • Works reasonably well with typical medical images 	<ul style="list-style-type: none"> • Performance is distribution dependent • Performance reduces with increasing number of bins • Poor performance for images with locally uniform intensities
2 - Synchronization-free parallelization	<ul style="list-style-type: none"> • Exact computation of histograms • No need for synchronization or atomic operations • Performance is virtually independent of the distribution • Suitable for degenerate or close to degenerate distributions for low to mid bin ranges 	<ul style="list-style-type: none"> • Performance reduces with increasing number of bins • Performs worse than method 1 for most distributions
3 - Approximate histogram	<ul style="list-style-type: none"> • No need for synchronization or atomic operations • Performance is virtually independent of the distribution • Performance is moderately affected by increasing number of bins 	<ul style="list-style-type: none"> • Computes only an approximation of the histogram • Accuracy of approximations is inversely related to the number of warps
4 - Bitonic sort and count	<ul style="list-style-type: none"> • Exact computation of histograms • No need for synchronization or atomic operations • Performance is virtually independent of the distribution • Performance is almost unaffected with increasing number of bins • Best performance for 10,000 and more bins 	<ul style="list-style-type: none"> • Lower performance for low and mid bin ranges

5.3.6 Computation of Joint Histograms

So far we have focused on computation of 1D histograms. We note that 2D or joint histogram computation can be cast as a 1D histogram computation problem, as follows.

Consider two real-valued sequences of equal size $s_1 : \{a_1, a_2, \dots, a_n\}$ and $s_2 : \{b_1, b_2, \dots, b_n\}$. Without loss of generality, we assume that sequence values are between 0 and 1. We would like to determine a joint histogram with $B_1 \times B_2$ bins, where B_1 and B_2 are the number of bins for s_1 and s_2 , respectively. The joint histogram computation can be reformulated as a marginal histogram computation with $B = B_1 \times B_2$ bins for a new sequence s given by

$$s : \left\{ \frac{\lfloor a_1(B_1 - 1) \rfloor + \lfloor b_1(B_2 - 1) \rfloor B_1}{B - 1}, \dots, \frac{\lfloor a_n(B_1 - 1) \rfloor + \lfloor b_n(B_2 - 1) \rfloor B_1}{B - 1} \right\},$$

$$B_1 B_2 > 1. \quad (5.7)$$

This simply allows us to reuse 1D histogram code for joint histogram computations.

5.4 Parallel Registration

In this section, we discuss fine-grained parallelization of MI-based registration on the GPU by incorporating various parallel histogram computation methods and parallelizing transformations in our registration framework.

On a CPU, registration method's execution time is typically dominated by the transformation function. GPUs, however, are specifically designed to perform geometric transformations. Transformations for individual elements are independent and can be efficiently parallelized. Geometric transformations (regardless of their type) require some sort of interpolation that involves adjacent voxels in a cubic region of memory. Standard computer architectures are designed to optimize sequential memory access through their caching mechanism. This does not fully benefit 3D interpolations over a cubic mesh. Modern GPUs, on the other hand, support a 3D texture addressing mode that takes the geometric locality into account for caching purposes. This greatly improves the efficiency of transformations on GPUs. Fig. 5.12 depicts the throughput of affine transformations given in 1 million voxels per second for CPU- and GPU-based implementations where linear interpolation is used. The figure demonstrates the superior performance of GPUs for geometric transformations.

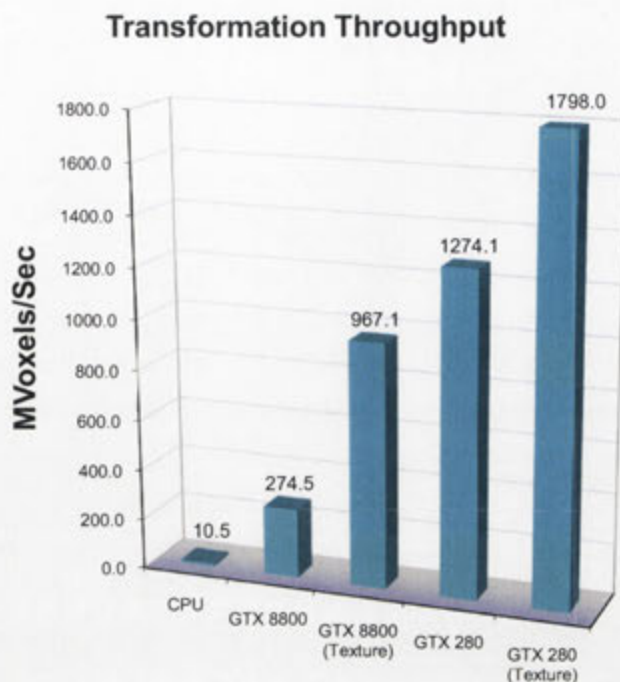


Figure 5.12: Performance of affine transformations on the CPU and on the GPU with and without use of textures. GPU implementation with textures performs significantly better.

5.4.1 MI Computation

As noted in Chapter 2, mutual information of two random variables is given by

$$I(X; Y) = \sum_x \sum_y p(x, y) \log \frac{p(x, y)}{p(x)p(y)}, \quad (5.8)$$

where $p(x, y)$ is the joint pmf of the two random variables. This is all we need for MI computation (marginal pmfs can be derived from the joint pmf).

There are various methods for estimating the joint pmf of image intensities. We divide them into two broad categories of *histogram-based* and *histogram-free* methods. Histogram-based methods are by far the more common and include direct estimation of the joint probability from a normalized histogram, partial volume histogram [18, 88], uniform volume histogram [21], and various Parzen window histogram computation methods [89, 90].

The most straightforward histogram-based method is to transform the moving image and interpolate image intensities in the transformed image from the original using linear interpolation or similar methods and then compute the joint

histogram of intensities from corresponding pairs of points in the fixed and moving images. However, MI computed in this manner, does not smoothly vary with the registration parameters and may not be suitable for gradient-descent based optimization methods. Other histogram-based methods such as [88] and [90] result in MI functions that smoothly vary with the change in registration parameters. These methods also provide closed-form solutions for MI derivatives which allows the use of gradient-based optimization methods (type A or B).

The basic sort and count algorithm can be used for direct computation of a standard joint histogram as discussed in the previous section. Our main aim is to demonstrate the computational advantage of our method and we have demonstrated that using direct histogram computation in our experiments in Section 5.4.3. However, we note that the presented methods can also be used for other histogram-based MI computation methods. These method typically require multiple non-integer histogram updates per joint sample of fixed and moving images. These histogram computation algorithms can be accommodated with minor changes to the basic algorithm.

As an example, we briefly discuss PV histogram method and the changes to the sort and count method needed for its computation.

Under a transformation T , a point such as \mathbf{y} in the fixed image F will correspond to $\mathbf{x} = T\mathbf{y}$ in the moving image M . However, since the coordinates of \mathbf{x} are generally non-integral, the intensity of the transformed point has to be interpolated from the neighboring points with integer coordinates. In a standard joint histogram, once the intensity of \mathbf{x} is computed, the histogram bin corresponding to $(M(\mathbf{x}), F(\mathbf{y}))$ is incremented by 1. In a partial volume histogram, however, \mathbf{x} contributes to multiple histogram bins associated with its neighboring grid points. The contributions depend on the distance of \mathbf{x} from its neighbors given by

$$P_v(\mathbf{x}, \mathbf{z}) = \prod_{i=1}^d (1 - |x_i - z_i|), \quad (5.9)$$

where \mathbf{z} is a neighboring point of \mathbf{x} , and d is the number of dimensions. The partial volume histogram updates the histogram bin corresponding to $(M(\mathbf{z}), F(\mathbf{y}))$ by $P_v(\mathbf{x}, \mathbf{z})$. For 3D images, partial volume histogram involves 8, generally non-integral, updates to the histogram.

The following minor changes allow the basic sort and count algorithm to be used for partial volume histogram computation:

1. Bin counters and the histogram data types are defined as floating point types.

2. The program adds multiple bins for each pair of points to the sort and count queue.
3. Bin counters are initialized to corresponding partial volume contributions.

Computation of MI derivatives using the partial volume histogram method also translates into computation of an appropriate histogram for each derivative [88], where the above method can be used again.

5.4.2 Experiments: Approximate Histogram

We integrated the histogram approximation method in our registration framework. In this section, we demonstrate that the method is useful and reasonably accurate for practical registration problems. We show that the MI functions derived using the approximate method are well-behaved and can be used to correctly determine the misalignment between the images.

Fig. 5.13 shows several MI functions based on the approximate histogram and the exact histogram for two MR images of the brain (MR-T1 and MR-T2). MI functions based on the approximate histogram are well-behaved, smooth and correctly identify the alignment. We are not concerned with the absolute difference between MI values computed by different methods. As it is the shape of the MI function which is important in registration applications.

We note that in Fig. 5.13, the MI functions with higher number of threads (more down-sampling) are slightly less smooth, however, they are smooth enough for optimization purposes. The smoothness of the MI function is related to the size of the input data. Larger data-sets tend to remain smoother for larger number of threads. However, we emphasize that a lower number of threads is recommended for smaller data-sets.

We finally show results of using this method for registration of 3D images (MR T1, T2 and PD). The images have approximately 7×10^6 voxels with a voxel size of 1 mm^3 . The misalignment between the images (gold) and the resulting registration parameters for standard MI calculation (single core CPU) and the approximate method (GPU: GTX 8800) are shown in Table 5.3. t_x , t_y and t_z are translation parameters along the x , y and z -axis, respectively. Rotations around x , y and z -axis are shown by α , β , and γ , respectively. We used 256 threads for the approximate histogram method on the GPU. The simplex method was used for the optimizations.

The registration results of the GPU method are comparable with the CPU implementation. Both methods converged with around 200 iterations but the GPU-

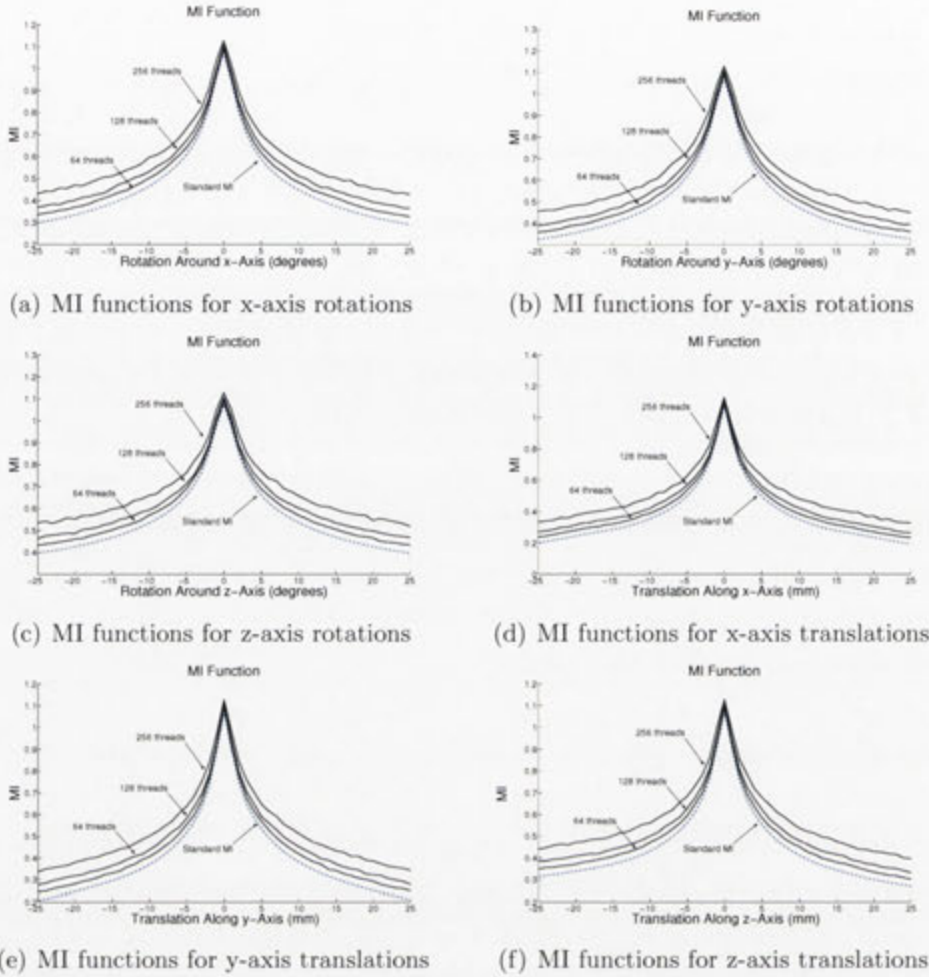


Figure 5.13: Comparison of MI functions for various misalignments. The dotted graph shows the standard MI function for two multi-modal images of the brain. The solid graphs show the results for the approximate MI method. The graphs show that the MI function is well-behaved and suitable for registration.

based registration on a GTX 8800 is around 45-65 times more efficient. We also ran the registrations on a single core of GTX 295 which performed the registrations in 2.25, 2.49, and 2.29 seconds for T1-T2, T1-PD and T2-PD, respectively. This is around 80-110 times faster than the standard CPU implementation. The standard registration on the CPU was reasonably optimized, albeit without the use of SSE instruction set⁸.

⁸For this reason, there is quite a bit of room for further optimization of the CPU-based implementation. Hence, GPU's performance gain compared to a single core CPU should be taken as indicative only.

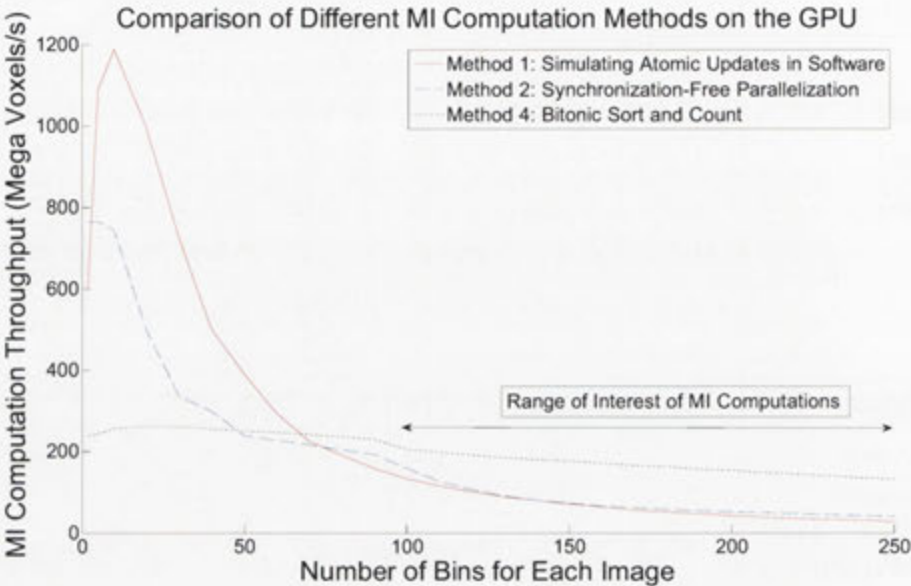
Table 5.3: Registration results for approximate histogram computation

Modalities		T1-T2		T1-PD		T2-PD	
	Gold	CPU	GPU	CPU	GPU	CPU	GPU
t_x (mm)	5.00	5.00	5.00	5.01	4.97	5.01	5.00
t_y (mm)	-10.00	-10.00	-9.98	-10.01	-10.00	-10.01	-10.00
t_z (mm)	-5.00	-5.03	-4.98	-5.01	-5.11	-4.98	-4.99
α	15.00°	15.01°	15.00°	15.01°	15.00°	15.00°	15.02°
β	-10.00°	-9.99°	-10.01°	-10.00°	-10.06°	-9.99°	-10.00°
γ	10.00°	10.03°	10.03°	10.02°	10.01°	10.00°	9.97°
Iterations		224	274	177	305	217	286
Time (sec)		251	3.89	198	4.34	243	4.07
Normalized Execution Time (ms/MVoxel/itr)		158	2.00	158	2.01	158	2.01

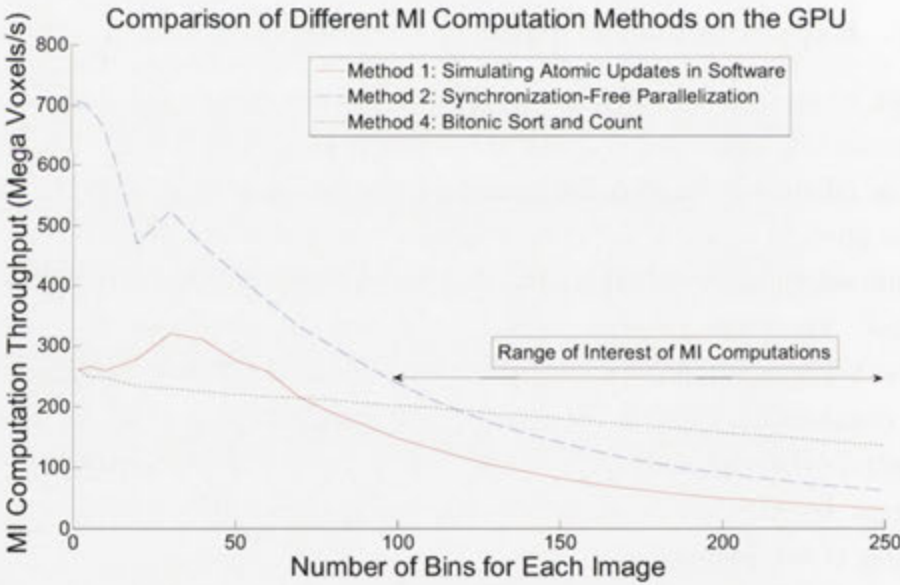
5.4.3 Experiments: Bitonic Sort and Count

We start by integrating different histogram computation methods in our registration framework. Fig. 5.14 displays the throughput of MI computation methods based on different histogram algorithms for a wide range of bins. Note that the x-axis is given in terms of the number of bins used per image in MI computation. The equivalent number of 1D bins needed for corresponding histogram algorithms is squared. The y-axis gives the throughput in mega voxels per second for MI computation between two images. Anywhere between 64-256 bins per image are used for MI computation with bins in excess of 100 being more common. As the images demonstrate the sort and count method gives the best performance for most of this range. For this reason, we use this method for registration experiments in the remaining of this section.

We used Vanderbilt database of brain images (patients 1-9) for this set of experiments. The database contains MR-T1, MR-T2, MR-PD, CT and PET images of real patients. In total, we performed 47 CT to MR registrations and 41 PET to MR registrations for each experiment. Fig. 5.15 shows a sample MR-T1 and CT image from the Vanderbilt database. Even though our focus is to demonstrate efficiency of our method, we also provide target registration errors (TRE) for completeness. Accuracy is measured at multiple volumes of interests (VOIs) in the brain that are of neurological significance. The TREs are computed against VOIs registered



(a) MI for two random variables with a uniform distribution



(b) MI for two 3D medical images from the Vanderbilt database

Figure 5.14: Performance of different MI computation methods on the GPU. For a range of bins that is of most interest for MI applications, bitonic sort and count performs best. Also note this method's independence from the data's underlying distribution.

Table 5.4: Accuracy comparison (median errors in mm)

Modality	MIRIT	Our Method (enh. Powell)	Our Method (std. Powell)
CT to T1	4.73	1.48	1.35
CT to T2	5.30	1.44	1.50
CT to PD	3.50	1.48	1.42
PET to T1	6.33	3.99	4.08
PET to T2	7.33	3.38	3.20
PET to PD	3.19	3.78	3.77

using the gold standard transformation which were obtained in the original study by using fiducial markers [1].

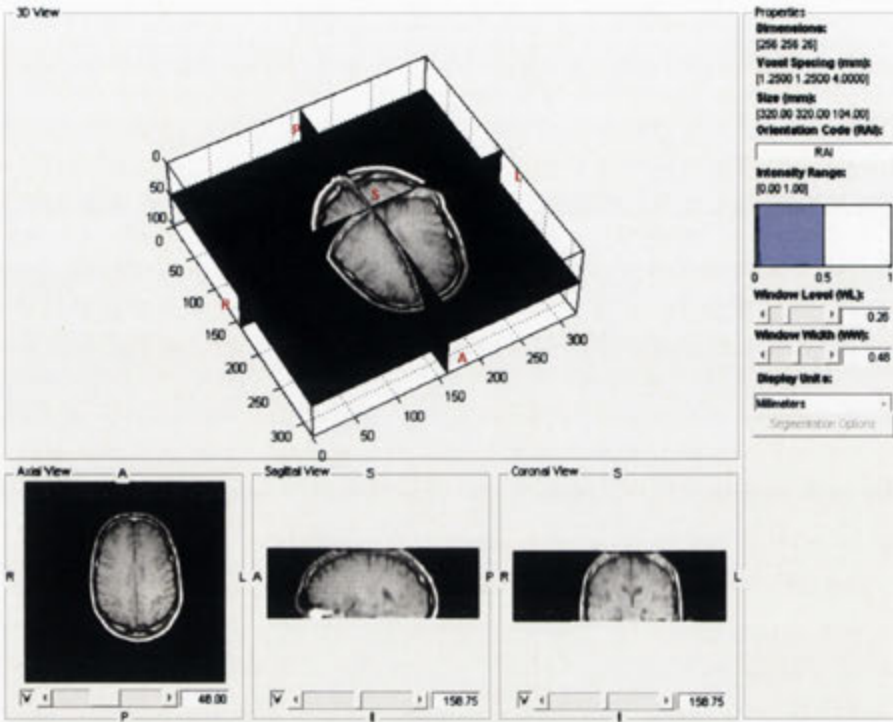
The Vanderbilt database is used for evaluation of rigid 3D registrations. While we limit our experiments to rigid registrations, we note that our GPU-based implementation is readily capable of performing affine registrations. In addition, the MI computation algorithm can be adapted for non-rigid registrations.

We compared performance and accuracy of our method against MIRIT⁹, a CPU implementation of MI-based registration by Maes et al. [18]. We ran MIRIT with recommended options except for the *partial volume* option [18]. We found the accuracy of the registrations for this particular data-set were actually better without this option. Both methods use Powell optimization algorithm. Our registrations were performed using the enhanced Powell method (refer to Section 3.2.3) with a resolution of 0.02 mm for translation parameters and 0.05° for rotation parameters. The results with standard Powell (no resolution parameters) are also included to demonstrate that the accuracy of registrations is not noticeably affected as a result of introducing the resolution parameters, while the performance is considerably improved.

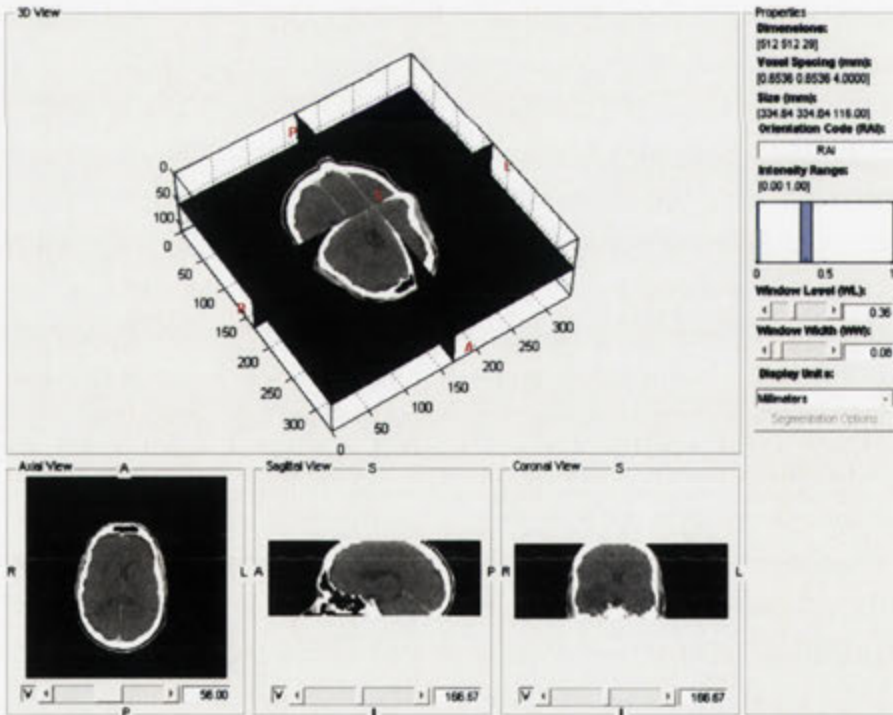
Fig. 5.16 shows the performance of bitonic sort and count method for 3D-3D registrations. On average, we were able to register each image pair in less than one second which represents more than 50 times speedup compared to MIRIT running on a CPU. We also note that by using the enhanced Powell method the performance of the GPU-based method can be improved by around 300%.

Table 5.4 shows the median TRE for MIRIT, our method with enhanced Powell, and our method with standard Powell optimization. Our method with or without resolution parameters, achieves sub-voxel accuracy and even outperforms MIRIT. MIRIT failed to converge for a few registrations. MIRIT would be able to reach

⁹Multi-modality Image Registration using Information Theory



(a) An MR-T1 Image



(b) A CT Image

Figure 5.15: Sample images from the Vanderbilt database displayed in a graphical user interface (GUI) developed under MATLAB[®] as part of this thesis.

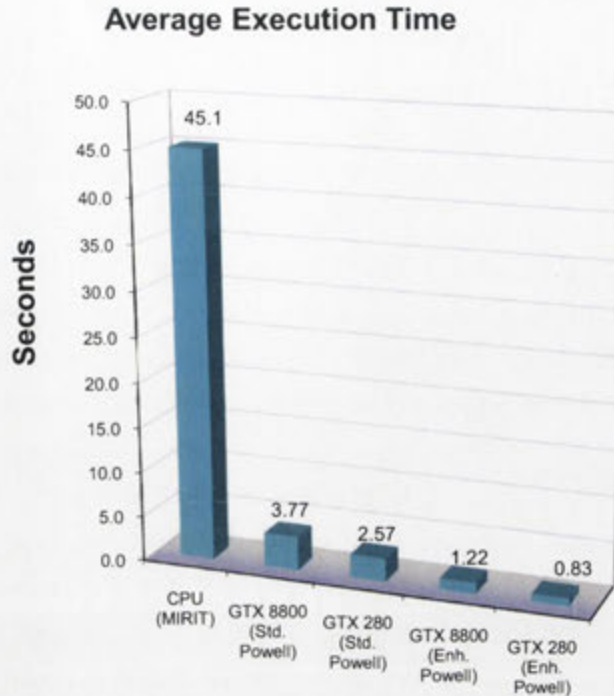


Figure 5.16: Average execution speed for pairs of images from the Vanderbilt dataset. Our method on GTX 280 is more than 50 times faster than MIRIT.

convergence with a different set of parameters for these cases, however, this would result in failed registration of other successful cases. We set to compare the methods in a fully automatic setting and it was only fair to run both with a single set of parameters. We note that, where convergence was achieved by both methods the TREs were more or less comparable.

5.4.4 Fast Deformable Registration

In this section, we look at improving the performance of B-spline deformable registration using the methods described earlier. The implementation is based on [7] and we implement both MI-based and SSD-based cost functions. We also developed a type D gradient descent algorithm using the *conjugate gradient* optimization method [22] with Brent's line minimization. The gradient of the cost function is computed numerically using forward finite differences.

An important benefit of B-splines as the basis for deformable registration is their computational efficiency. This stems from the property that when a single control point in a B-spline control grid is moved, this only affects the position of points in the local vicinity of that control point. The size of the affected region is at most

equivalent to a volume covered by a mesh of 5 neighboring control points. For a control point $C_{i,j,k}$ this is the volume enclosed between $C_{i-2,j-2,k-2}$ and $C_{i+2,j+2,k+2}$. This is particularly important in computation of the gradient of a measure using finite differences. Each partial derivative of the measure w.r.t. a coordinate of a control point affects a sub-volume in the image. Assuming a uniformly distributed control mesh of size $n_x \times n_y \times n_z$, and noting that the image volume is covered by $(n_x - 2) \times (n_y - 2) \times (n_z - 2)$ control points (there are 2 control points outside the image volume in each dimension); the control points divide the image into $(n_x - 3) \times (n_y - 3) \times (n_z - 3)$ blocks, hence the computation of a single partial derivative is approximately $4 \times 4 \times 4 / ((n_x - 3) \times (n_y - 3) \times (n_z - 3))$ times that of the cost function itself. The gradient consists of $3 \times n_x \times n_y \times n_z$ partial derivatives, therefore the computational complexity of the gradient is only $192 \times n_x \times n_y \times n_z / ((n_x - 3)(n_y - 3)(n_z - 3))$ times more expensive than computation of the cost function itself, if properly implemented. This involves storing partial transformation and measure computation results for each block enclosed by adjacent control grids. During the computation of the gradient, we only need to evaluate blocks that are affected by the movement of a single control point and update the cost function accordingly. Also note that the complexity of the gradient computation in this manner is independent of the size of the control grid for a dense mesh. For certain types of images, such as brain images, that are fully enclosed within the volume, we can further improve the computational efficiency of the method by not evaluating the partial derivatives of control points that fall outside the boundaries of the image. This further improves the computation time of the gradient to approximately $192(n_x - 2)(n_y - 2)(n_z - 2) / ((n_x - 3)(n_y - 3)(n_z - 3))$ times of a single cost function evaluation.

The B-spline transformation can be efficiently implemented on the GPU by mapping the moving image to a 3D texture and storing the coordinates of the control grid in the GPU's constant memory. We use a multi-resolution multi-grid strategy for registrations. Table 5.5 shows the performance of our implementation for sample images of a given size and the normalized performance in ms/MVoxels/itr of the gradient descent algorithm, where, each iterations involves the computation of the cost function's gradient together with several cost function calls during the line minimization stage. The performance results are given for the finest level of a multi-resolution pyramid and a grid size of $7 \times 7 \times 7$ run on a single core of a GTX 295.

The outcome of a sample registration is shown in Fig. 5.17. The moving image is a distorted version of the fixed image obtained by applying a rather large (and

Table 5.5: Performance of B-spline registrations

Size of Images	Measure	Cost	Gradient	Normalized Perf.
$181 \times 217 \times 180$	SSD	27 ms	3,652 ms	573 ms/MVoxels/itr
	MI	70 ms	11,113 ms	1,699 ms/MVoxels/itr

deliberately exaggerated) sinusoid deformation field given by

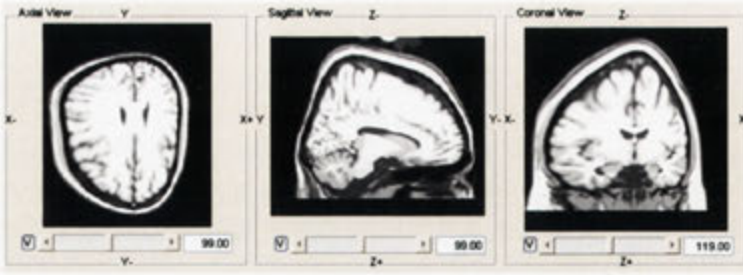
$$\mathbf{d}(x, y, z) = \left[10 \cos\left(\frac{x\pi}{360}\right) \ 10 \sin\left(\frac{y\pi}{360}\right) \ 10 \cos\left(\frac{z\pi}{360}\right) \right]^T. \quad (5.10)$$

We use a 3 level multi-resolution combined with a multi-grid control mesh to register the images and recover the transformation field. The size of the control mesh is varied from coarse to fine ($5 \times 5 \times 5$ to $11 \times 11 \times 11$) at each resolution level. Fig. 5.17(c) shows the resulting transformed image after the recovered deformation field is applied to the moving image. Note that the transformed image and the fixed image are visually very close. This can also be seen from the difference images. Fig. 5.17(d) shows the difference between the moving and fixed images prior to registration where large mismatches between the two images can be observed. Fig. 5.17(e) shows the difference between the fixed image and the transformed moving image after the completion of the registration where the mismatch between the images is minimal and hardly visible. A typical multi-resolution multi-grid registration takes anywhere from 3-8 minutes to complete on a single core of a GTX 295.

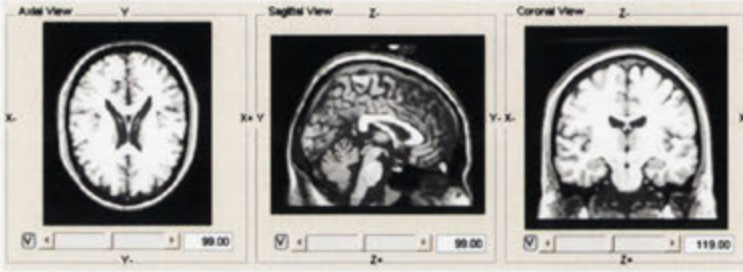
5.5 Discussion

The methods presented in this chapter, highlight some of the challenges, trade-offs and benefits in rethinking existing algorithms for a massively multi-threaded architecture, such as CUDA. We demonstrated that adaptation of existing applications to massively multi-threaded architectures involves re-inventing the algorithms rather than a simple port of existing ones. The sheer number of parallel threads involved, challenges traditional algorithm and software design practices. Some standard tools such as synchronization primitives become even undesirable. We argue that for massively multiprocessing architectures, the best synchronization, may be no synchronization at all.

A properly designed application on the GPU can achieve significant performance gains compared to a standard CPU implementation while offering a lower cost per



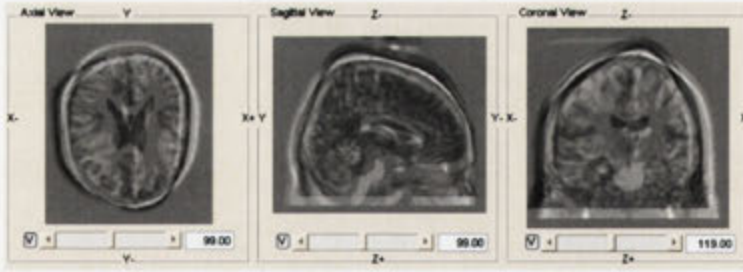
(a) Moving image



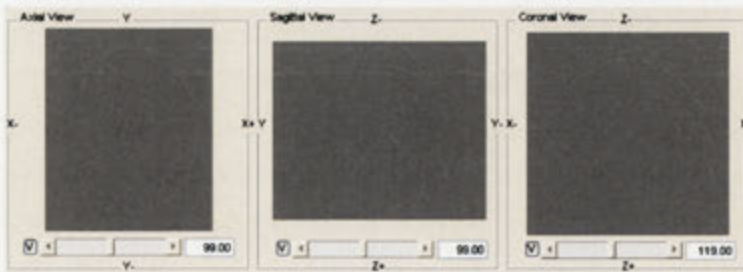
(b) Fixed image



(c) Transformed image



(d) The difference between fixed and moving images



(e) The difference between fixed and transformed images

Figure 5.17: B-spline registration on the GPU: The deformation between (a) moving and (b) fixed images is recovered and applied to the moving image. The final result is shown as (c), the transformed image. The difference between moving and fixed images is shown in (d). The difference after registration is shown in (e), where almost no misalignment between the images can be seen.

TFLOP of computing capacity, a lower footprint and a higher performance per Watt. The advent of this new generation of low-cost high performance computing platforms presents both numerous opportunities and challenges.

Medical image analysis is one of many computationally-intensive disciplines that stands to benefit greatly from this evolution of high performance computing. The capacity is now here to tackle larger-sized problems, that may have been considered intractable only a few years ago, on clusters of GPUs. The extra computational capacity can also be spent towards development of more robust tools which require less supervision to complete tasks or to adapt preoperative tools for intraoperative applications or invent new ones. These require a more coordinated effort by the medical imaging community to redesign a large set of existing and fundamental algorithms in areas such as registration, segmentation and modeling. A feat which many not be possible without a more enthusiastic embracement of high performance computing within the field. We hope that existing efforts pave the way for the conception of a *computational medical imaging* community in much the same way that computational physics, chemistry and astronomy have grown out of their respective disciplines.

Chapter 6

Real-time Simulation and Visualization of Ultrasound

6.1 Introduction

Ultrasound as an imaging modality is desirable from many perspectives: (a) it is real-time with a high temporal resolution, (b) it is risk-free (radiation-free and non-hazardous), (c) the ultrasound devices are relatively cheap, (d) ultrasound devices are portable and relatively small, and (e) ultrasound probes can be used to target small tissue interfaces in endoscopic, laparoscopic and intravascular applications. The main drawback, however, is the quality of the acquired images and a low signal to noise ratio (SNR), which makes navigation and interpretation of the acquired images, particularly challenging. Ultrasound simulation systems have been shown to improve the performance and skills of users, significantly (e.g. see [91]). This is due to the fact that the trainees can practice localization and acquisition of ultrasound without the time-constraints imposed by such practice on the patients and can also access a variety of cases which have been collected and stored in the simulation system's database over time.

A number of systems including commercial products are available for ultrasound training of medical students and staff (e.g. [92–100]). These systems allow navigation with a virtual probe within the space of pre-recorded ultrasound images. The acquisition protocol is typically 3D freehand ultrasound with a compounding stage where 2D ultrasound images are combined to create a 3D volume or straight 3D ultrasound. At run-time, during training sessions, the position and orientation of the virtual probe is tracked and the relevant ultrasound planes are re-sliced from the previously computed volumes. Technically, these systems simulate the ultrasound acquisition rather than the ultrasound phenomenon itself.

One common problem with traditional ultrasound simulation systems (e.g. [92–100]) is that the simulation is realistic as long as the operator remains within close vicinity of originally acquired positions and orientations. As the probe is navigated further away from the acquisition positions, the images become less realistic, since view-dependent ultrasound effects are no longer accurately represented. The acquisition protocol is also complicated and requires the volume of interest to be imaged from various positions and not to contain view-dependent artifacts such as shadowing, and the effect of a fixed gain and focus. Then there is, of course, the issue of compounding the images and accumulated errors due to mis-registration and accumulation of intensity values with varying intensities due to view-dependent artifacts.

More recently, simulation of ultrasound from CT volumes has attracted interest. Authors in [101] briefly discuss simulation of ultrasound from CT volumes without providing much detail on their ultrasound modeling. In [102], the authors discuss a system for ultrasound guided needle insertion and use CT images for patient specific training. Ultrasound simulation using a simple ray-based modeling of ultrasound is used in [103] for registration of ultrasound images to CT volumes. In [104], we proposed an enhanced modeling of ultrasound, which results in more realistic ultrasound simulations from CT images suitable for ultrasound training.

Use of CT images as the basis for simulations not only avoids the aforementioned drawbacks but also has the advantage of allowing for patient specific simulations, ease of navigation for novice users as they can practice ultrasound navigation with the help of corresponding CT information (this extra assistance is obviously turned off at later stages of training). It also provides easier access to raw data for simulation, as CT images are routinely acquired for diagnostic and planning and the acquisition protocol is not complicated and is streamlined.

In this chapter, we use the term ultrasound *simulation* exclusively in the context of creating fully synthetic ultrasound images. We synthesize the ultrasound image from a fixed-view scattering image and a view-dependent reflection image. The scattering image is generated off-line using Field II [105, 106] and the reflection image is based on a simple model for acoustic wave propagation in a piecewise homogenous medium and is generated in real-time. Both images are derived from a 3D CT scan of the region of interest.

Fully synthetic simulation of ultrasound has been previously investigated by Jensen et al. [105–107] based on an acoustic wave-propagation model and using the concept of spatial impulse response [108, 109] which is implemented in a program called Field II [105]. The program can be used to simulate any linear ultrasound

system with single or multi-element transducers, any given apodization, focusing, pulse excitation scheme and aperture geometry [110]. The program requires location and strength of scatterers as input and gives best results with carefully designed and synthetically generated scattering patterns. As such, the program is mostly used to determine the effects of various parameters on transducer design. Additionally, the simulations for even a single B-mode image take an extremely long time and need to be parallelized (the execution time for a B-mode image with 128 RF scan lines and 1,000,000 point scatterers is in the order of 2 days on a single CPU), which makes it impractical for real-time simulation and in training applications.

In this chapter, we present a framework for ray-based simulation of ultrasound including an efficient modeling of ultrasound and its implementation on the GPU in a simulation and visualization software. The framework accommodates ultrasound simulations with varying degrees of complexity and is fast enough to produce interactive simulation and visualization for training and registration purposes.

6.2 Method

6.2.1 A Simple Acoustic Model for Ultrasound

We develop a simple acoustic model that can be used in real-time for simulation of large-scale reflections, attenuation due to reflections, effect of a finite beam-width, and view-dependent shadow and occlusion effects in an ultrasound image.

Reflection: A sound beam traveling through a piecewise homogenous medium is partially reflected at the interface of two media with differing *acoustic impedances*. This impedance mismatch is the primary mechanism that allows visualization with ultrasound. The acoustic impedance is defined by $Z = \rho c$, where ρ is the density of the medium and c is the speed of sound. The ratio of reflected energy to incident energy is called the *reflection coefficient*, α_R , and is given by

$$\alpha_R = \left(\frac{Z_2 - Z_1}{Z_2 + Z_1} \right)^2, \quad (6.1)$$

at the interface of the two media with acoustic impedances Z_1 and Z_2 [111]. The remaining energy that passes through the interface is characterized by the transmission coefficient $\alpha_T = 1 - \alpha_R$.

Refraction: The transmitted beam, is often refracted due to a change in the speed of sound from one medium to the other. Refraction of sound beams obeys Snell's law: $\sin \theta_i / \sin \theta_r = c_1 / c_2$, where θ_i and θ_r are the angle of incidence and refraction and c_1 and c_2 are the speed of sound in the two media. For most soft tissue interfaces, the effect of refraction particularly close to the direction of the surface normal is small. This is due to little variation in the speed of sound among most soft tissue types with the exception of fat, where deviations up to 20° due to refraction can be experienced. In our model, we ignore the effect of refraction and assume that $\theta_r = \theta_i$. A more comprehensive model based on ray-tracing and tissue labeling is required in order to take refraction into account.

Lambertian Scattering: Reflection of sound beams at an interface is the main interaction of interest to us. The reflection is typically non-specular and subject to scattering. The intensity of the scattered signal (from a receiver's point of view) depends on the angle of incidence and is maximal for a beam normal to the interface and approaches zero as the incident angle approaches 90° . This effect can be described by Lambertian scattering¹. The intensity of the reflected signal at a point \mathbf{x} on the interface of the media depends on the angle of incidence and can be written as

$$R(\mathbf{x}) = \alpha_R(\mathbf{x}) I_i(\mathbf{x}) |\mathbf{r}(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x})|, \quad (6.2)$$

where $I_i(\cdot)$ is the intensity of the incident beam at the interface, \mathbf{r} is the unit vector in the direction of the beam, \mathbf{n} is the surface normal, $|\cdot|$ is the absolute value operator, and $R(\cdot)$ is the intensity of the reflected signal. According to the Lambertian scattering model, the intensity of the reflected signal, perceived by an arbitrary viewer, is independent of the view angle and only depends on the angle of incidence. This is of course provided that no attenuation occurs in the return path between the point of reflection and the observer. If the initial intensity of the transmitted signal is shown by I_0 , the attenuation coefficient at point \mathbf{x} is given by $I_i(\mathbf{x})/I_0$. Since the reflected signal travels back through the same attenuating medium (ignoring any refraction), the intensity of the signal as sensed by the receiver, $I_r(\mathbf{x})$, is attenuated by the same coefficient as in the forward path and

¹We use a Lambertian model for its simplicity. A more appropriate model for ultrasound scattering is Rayleigh scattering, since the interface dimensions are much smaller than a wavelength [111]. This is the subject of further investigation.

can be written as

$$I_r(\mathbf{x}) \propto \alpha_R(\mathbf{x}) \frac{I_i^2(\mathbf{x})}{I_0} |\mathbf{r}(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x})|. \quad (6.3)$$

Effect of Beam Width: The aperture of an ultrasound transducer consists of a number of acoustic elements. Typically a group of adjacent elements are actively sending and receiving acoustic signals while others are turned off (see Fig. 6.1a). Using a group of active elements produces a deeper *near field* and a less diverging *far field* compared to a single element acting alone [111]. The active aperture is electronically shifted along the aperture to cover the entire field of view. This results in the transmitted signal from a single element to be partially received by adjacent elements. One novel aspect of the present work is modeling this effect. The reflected signal is integrated along the active wavefront at a specified depth using a suitable window function which results in a more realistic reflection image (see Fig. 6.2). For a linear array transducer we can write

$$I_r(x, y) \propto \int_{x-\ell}^{x+\ell} \alpha_R(u, y) \frac{I_i^2(u, y)}{I_0} |\mathbf{r}(u, y) \cdot \mathbf{n}(u, y)| \omega(u) du, \quad (6.4)$$

where $\omega(\cdot)$ is the window function, and ℓ is length of the active aperture, given by $\ell = n_a(w_e + s_e)$, where n_a is the number of active elements, w_e is the width of each element, and s_e is the spacing between adjacent elements. If the active aperture is moved at a constant frequency, f_a , each element will be active for a period of time equal to $(2n_a - 1)/f_a$. Let us consider the m^{th} element in the transducer: during its operating interval, it will receive reflections due to operation of elements in the range $m - n_a + 1$ to $m + n_a - 1$. The amount of reflected signal due to an element $i \in [m - n_a + 1, m + n_a - 1]$ is proportional to the amount of time when i and m are both turned on, which is given by

$$t(i) = \frac{2n_a - 1}{f_a} \left(1 - \left| \frac{i - m}{n_a} \right| \right). \quad (6.5)$$

This results in a triangular window function $\omega(\cdot)$ in (6.4). In our simulations, though, we used a Hann window to further suppress the contributions from elements that are farthest away.

Note that thanks to modeling the beam width effect, one can infer the direction of ultrasound beams by looking at the content of Fig. 6.2b and Fig. 6.2c.

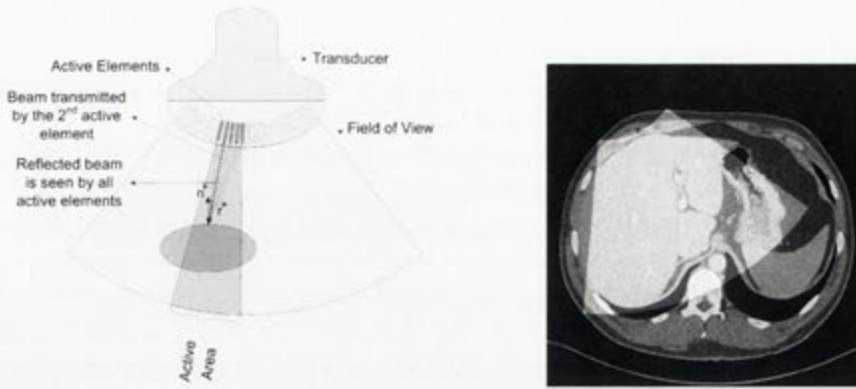


Figure 6.1: From left: (a) A convex array transducer with a multi-element active aperture, (b) Ultrasound field of view superimposed on the liver of a human subject.

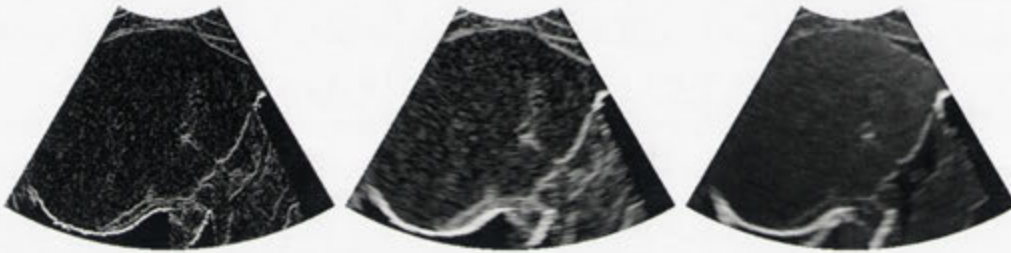


Figure 6.2: From left: (a) Reflection image with one active element, (b) Reflection image with multiple active elements, (c) Our simulated ultrasound (combined reflection and scattering images.). Note: images depict the region of interest shown in Fig. 6.1b and are log-compressed. Also notice shadowing on the right-hand side due to an air-tissue interface and in the middle-bottom of the image due to a bone-tissue interface.

6.2.2 Creating the Reflection Image

The reflection image simulates view-dependent ultrasonic effects due to reflection and attenuation of the signal. Tissue boundaries are emphasized in the image and shadows due to large impedance mismatches between tissue-bone and tissue-air interfaces are simulated (refer to Fig. 6.2).

We use a CT volume for real-time simulation of the reflection image. An edge volume, based on the method proposed in [112] with a Deriche filter, is computed from the CT image. Edge detection needs to be performed once, when the CT image is first loaded. Given a set of acquisition parameters and position information, a corresponding plane from the CT and edge volumes is extracted. The 3D edge volume removes the need for 2D edge detection at run-time and also provides better continuity as the probe is navigated.

The implementation of the algorithm for a linear transducer is straightforward.

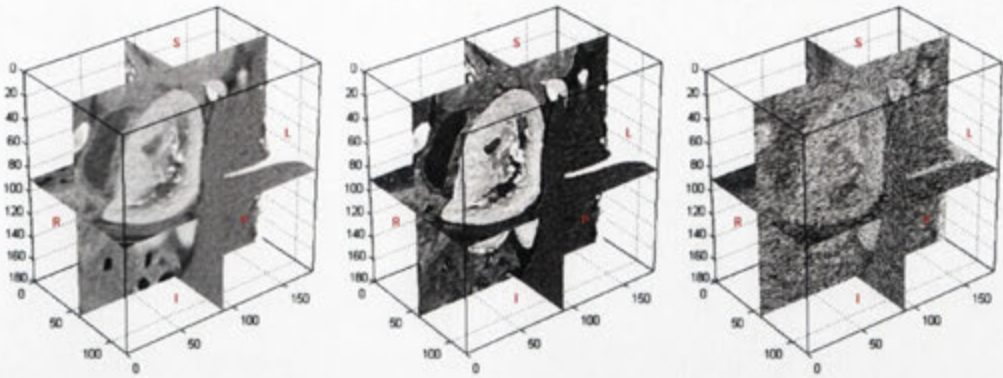


Figure 6.3: From left: (a) CT scan of the kidney of an animal subject, (b) Contrast-adjusted and edge-enhanced CT image used as a scattering map, (c) The scattering image generated by Field II.

Transmission and reflection coefficients and angles of incidence are calculated at the interface between every two media along an axial scan line. The interfaces are detected from the edge map. The acoustic impedances of the media, divided by each interface, are determined from the average intensity of the CT image along the scan line between the interface boundaries. The CT intensities for tissues are approximately proportional to the acoustic impedance for tissues [103] and can be used directly for calculation of the reflection coefficients in (6.1). This is not true for bone and air, so we also label the media as bone, air and tissue based on their CT Hounsfield intensities. On average bone-tissue interfaces reflect 43% and air-tissue interfaces reflect 99% of the incident beam [111].

The process of simulating linear array and convex array transducers are similar with the exception that for convex array transducers, the extracted CT slice has a fan-shaped field of view. We first warp the fan-shaped area to a rectangle using a Cartesian to polar transformation. This, in turn, transforms the convex transducer to a linear one and allows us to use (6.4) for both geometries.

6.2.3 Creating the Scattering Image

Realistic speckle patterns can be simulated using software packages such as Field II [105]. Simulations are based on the principles of linear acoustics and computation of the spatial impulse response [109]. Speckle is simulated by randomly placed scatterers with strength randomly chosen by Field II from a normal distribution. The mean of this distribution is location-dependent, and is provided as input to the simulator in the form of a scattering map, which gives the mean scatterer strength at all points within the volume of interest. A typical B-mode image requires anywhere from 200,000 to 1,000,000 point scatterers in order to create

a realistic speckle pattern. Scattering simulation in this way is computationally expensive. On a standard PC, with 1,000,000 point scatterers, simulation of a single ultrasound beam takes almost 20 minutes. This is nearly 2 days for a B-mode image with 128 RF scan lines. For this reason the scattering image is preprocessed from a single view and stored along with the CT image.

Direct use of a CT image as a scattering map results in a repetitive scattering pattern where hardly any structures are recognizable. We overcome this by using a contrast- and edge-enhanced image (shown in Fig. 6.3) as our scattering map. First the CT image undergoes affine contrast-stretching to maximize contrast while allowing no more than 5% of voxels to under- or overflow (saturate) the intensity range. The resulting scattering map is then further enhanced by emphasizing tissue or organ boundaries, which represent highly scattering areas. This is done using the previously calculated edge-map; detected edge points are set to maximum value in the scattering map. Fig. 6.3 shows 3-view images of the original CT image, the edge- and contrast-enhanced scattering map, and the resulting scattering image which exhibits speckle.

The 3D scattering volume is simulated slice-by-slice in the axial plane. A virtual linear array transducer operating at 7.5 MHz, is positioned along the left-right (LR)-axis of the slice. The speed of sound is assumed to be $c = 1540 \text{ m/s}$ resulting in a wavelength of $\lambda = 205.3 \mu\text{m}$. Width of each transducer element is set to λ . Kerf (spacing between elements) is set to 0.1λ . The aperture length is slightly longer than the image width. In our experiments, this results in an aperture with 504 elements. There are 64 active elements in the aperture. 128 RF scan lines were simulated per slice. The virtual transducer was designed to provide a realistic scattering image while requiring the least computation time possible. We refer the reader to [113] for more information on the meaning and effect of these parameters on the resulting simulation. Fig. 6.4 shows a volume rendered CT image of the abdomen with the corresponding scattering image.

We used a 20 CPU cluster to parallelize the computations for a $180 \times 120 \times 180$ pixels volume of interest, depicting the kidney of an animal subject, cropped from a larger CT image with a spacing of $0.55 \times 0.55 \times 0.60 \text{ mm}$. Four scatterers were introduced per voxel resulting in a total of 15,552,000 scatterers for the entire volume and a total simulation time of nearly 32 hours.

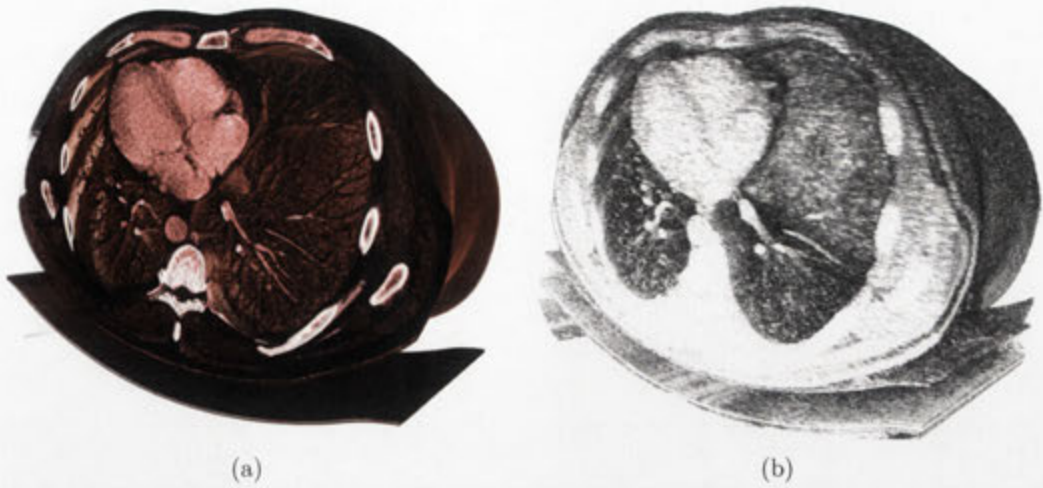


Figure 6.4: (a) Volume rendering of the CT image, (b) Volume rendering of the corresponding scattering image.

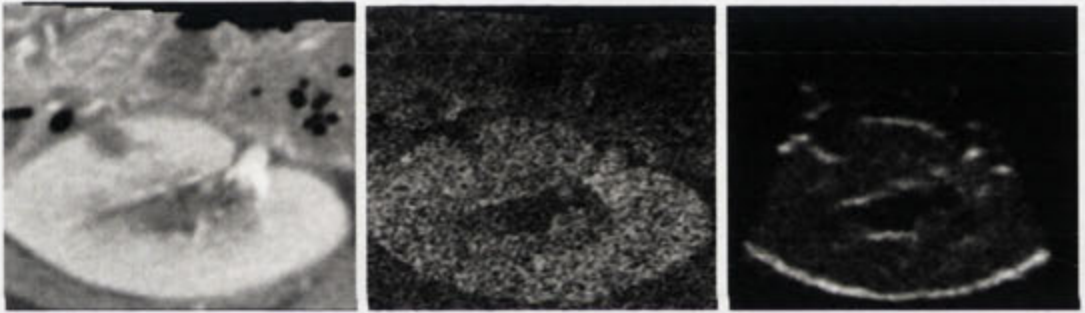


Figure 6.5: From left: (a) CT scan of an oblique plane within the CT volume, (b) Scattering image generated by Field II, (c) Our simulated ultrasound image: note the speckle pattern and strong reflection from kidney boundaries. The tumor on the top-left corner of the kidney can be easily identified.

6.2.4 Creating the Ultrasound Image

The final simulation is the result of combining the reflection and scattering images the following formula:

$$I_{us}(\mathbf{x}) = (G_{\sigma_1}(\mathbf{x}) * I_r(\mathbf{x}) + \alpha G_{\sigma_2}(\mathbf{x}) * \alpha_T(\mathbf{x}))I_s(\mathbf{x}), \quad (6.6)$$

where $I_{us}(\cdot)$ is the ultrasound image, $I_r(\cdot)$ is the reflection image, $I_s(\cdot)$ is the scattering image, α is a blending coefficient, and G is a Gaussian filter with 0 mean and adjustable standard deviation (σ_1 and σ_2) used to smooth the output of the image fusion process. Increasing α , results in a stronger speckle texture, while reducing it makes reflections more dominant. The blending parameters, α , σ_1 and σ_2 are adjusted by the operator for best viewing results.

The resulting image has a large dynamic range which far exceeds the dynamic range of the display and range of intensities that can be identified by the human eye. To reduce the dynamic range, we compress the signal using the following log-compression method

$$I_c(\mathbf{x}) = \begin{cases} 0, & I_{us}(\mathbf{x}) < \max\{I_{us}(\mathbf{x})\}10^{-\beta/10} \\ 10 \log_{10}(I_{us}(\mathbf{x})/\max\{I_{us}(\mathbf{x})\}) + \beta, & \text{otherwise} \end{cases}, \quad (6.7)$$

where β is the dynamic range of the compressed signal.

Fig. 6.5 shows an oblique plane within the CT volume and corresponding scattering and simulated ultrasound images. Note the highly reflective areas in the ultrasound around the kidney boundaries and vasculature and the realistic speckle pattern. The tumor can be easily located in the simulated ultrasound.

A C++ implementation of our algorithm (without any particular attention to optimization of the code e.g. use of SIMD instruction set) can generate images similar to Fig. 6.2c at 10-15 frames/sec. In the following section, we present a fast GPU-based method for simulation of ultrasound images and their visualization for interactive results.

6.3 Visualization and GPU-Accelerated Simulation of Ultrasound

In this section, we present a fast GPU-based method for simulation of ultrasound images from volumetric CT scans and their visualization. We show that the ultrasound simulation problem can be formulated as a ray casting problem. Ray casting has been extensively researched in the graphics community. Thus, we can benefit from the availability of an established framework for direct volume rendering and GPU-based image processing which has been up to recently largely developed using the graphics API. For this reason, despite the benefits offered by native general purpose programming platforms such as CUDA, we chose the graphics API for our GPU-based ultrasound simulation. Use of the graphics API also ensures that our implementation is relatively independent of the graphics hardware and can be run on a wider range of devices. The only requirement is a graphics card that supports *Shader Model 3.0* and *OpenGL 2.1*.

General purpose programming on the GPU (GPGPU) through the graphics API has been around for more than a decade. The programs can be implemented

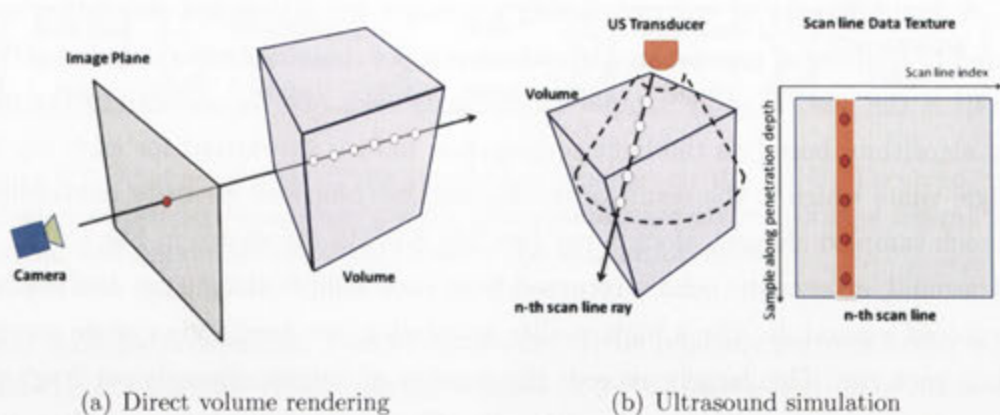


Figure 6.6: Difference of ray casting algorithms for direct volume rendering and ultrasound simulation (a) for DVR, multiple samples along a ray require a single storage (red dot in the image plane), (b) for ultrasound simulation, every sample along a ray requires a corresponding storage.

in programmable vertex and fragment shaders of the graphics pipeline mostly in an OpenGL or Direct3D environment using computer graphics shading languages, e.g. *OpenGL's built-in Shading Language (GLSL)*, *Microsofts High Level Shading Language (HLSL)*, or *NVIDIA's C for Graphics (Cg)*. The implementation of a non-graphics algorithm is therefore not straightforward as it has to be reformulated to be executed by the graphics pipeline. Additionally one has to circumvent the limitations imposed by the computer graphics environment, e.g. no random access writes (scatter writes) to the global GPU memory.

6.3.1 GPU-Accelerated Ultrasound Simulation

We formulate the ultrasound simulation problem on the GPU as a ray casting problem. The (virtual) ultrasound transducer is positioned within the space of the CT volume. For every transducer element, and depending on the geometry of the probe (i.e. linear or curvilinear), an ultrasound beam is cast and multiple rays are processed in parallel by the GPU. For each sample along a ray, equation (6.3) is computed inside a *fragment shader*. The results are stored as a measure of the acoustic intensity received by a transducer element from a point at a given depth in the anatomy along an ultrasound beam and displayed as an image.

The algorithm is implemented in C++, OpenGL, and GLSL. The OpenGL FramebufferObject (FBO) Extension² is employed to efficiently render to off-screen render targets (i.e. 2D and 3D textures in GPU memory).

²http://www.opengl.org/registry/specs/EXT/framebuffer_object.txt

A key difference of our ray casting algorithm for ultrasound simulation compared to traditional ray casting algorithms (e.g. for direct volume rendering (DVR) [114]) is the need to store sample values along each ray. In a standard ray casting algorithm, based on the light propagation model, the output for each ray is a single value which is the result of combining the color and intensity contribution of each sampled element along a ray (see Fig. 6.6(a)). As shown in Fig. 6.6(b), for ultrasound, an acoustic echo is returned from each sample along a ray and needs to be stored separately. For a high quality simulation, we need 256 or more samples along each ray. This largely exceeds the number of output channels per fragment. As such, we use a multi-pass algorithm for efficient implementation of ultrasound ray casting on the GPU.

Various data structures are allocated and loaded during the initialization stage and an optimal memory layout is determined. CT data, ultrasound ray start and end vectors are stored in textures. Ray start and end vectors are used to compute position of samples within the CT volume at each pass of the algorithm. Ray start and end vectors have to be re-initialized, every time that the user changes the orientation or position of the probe.

The ray casting algorithm is designed to be independent of the probe geometry and ultrasound dimensions. Scan line information is stored in 2D textures for both 2D and 3D ultrasound images. This is a major benefit and allows us to use the same algorithm for simulation of 2D, 3D, linear, curvilinear and freehand ultrasound. The original dimension and shape of the ultrasound image are restored in the scan conversion stage.

We need three render targets for storage of intermediate results and acoustic intensities. This is to store $I_i(\mathbf{x})$ and $I_r(\mathbf{x})$ (refer to Section 6.2.1). $I_i(\cdot)$ is calculated recursively

$$I_i(\mathbf{x}) = I_i(\mathbf{x} - \Delta\mathbf{d})(1 - \alpha_R(\mathbf{x} - \Delta\mathbf{d})), \quad (6.8)$$

where $\Delta\mathbf{d}$ is the incremental sampling vector along a given ray. Storage of $I_i(\cdot)$ scan line data requires two textures to avoid read/write conflicts and synchronization issues. The algorithms interleaves data read/writes for even/odd rows of the scan lines (ping-pong rendering). This is to ensure that all fragment shaders finish writing into row k , before starting row $k + 1$ which requires values of the previous row.

A practical consideration in allocating textures is the memory layout. GPUs typically have an upper-bound for the width and height of the textures. Regardless of the available memory, one cannot allocate a texture that exceeds the limit in



Figure 6.7: Various stages of the simulation pipeline on the GPU. The scan line and scan conversion stages are always executed. Other stages (orange boxes) are optional and only executed if required by the simulation model.

one or multiple dimensions. Performance-wise, GPUs typically perform better with square textures whose dimensions are a power of 2. We need a texture of size $n \times d$ for simulating an ultrasound with n transducer elements and d samples along each ray. This is not a problem for 2D ultrasound as the number of scan lines hardly exceeds 256. However, for 3D ultrasound the number of elements and as a result scan lines can easily exceed the limit. Therefore, the memory layout is optimized to be close to square and several tiles of scan lines are arranged within the texture, as needed.

6.3.2 The Simulation Pipeline

Our simulation pipeline as shown in Fig. 6.7 consists of five stages: the scan line traversal, pre-scan conversion, scan conversion, post-scan conversion and compositing stages.

The pre-scan conversion, post-scan conversion and compositing stages are optional and are executed if required by the underlying ultrasound model (e.g. the model in [103] utilizes the scan line and scan conversion stages only, while the more complex in [104] invokes all stages of the pipeline).

- *Scan Line Traversal Stage:* As the first stage of the pipeline, the 3D data-set is sampled along each scan line and the values are stored in a 2D texture. Each time probe-related parameters are varied by the user, scan line data has to be recomputed. For simulation of an ultrasound image with d samples (pixels) along each beam, the algorithm requires exactly d render passes. For simulation of 2D ultrasound, typically a single line primitive is executed at each pass. However, for 3D ultrasound or simulation of multiple 2D ultrasound images, where scan lines are tiled within the texture memory, we run m parallel line primitives, where m is the number of tiles. Running multiple line primitives typically provides a better utilization of the GPU resources. This means that our algorithm reaches its full capacity (throughput) for larger simulations (i.e. multiple 2D and 3D ultrasound).

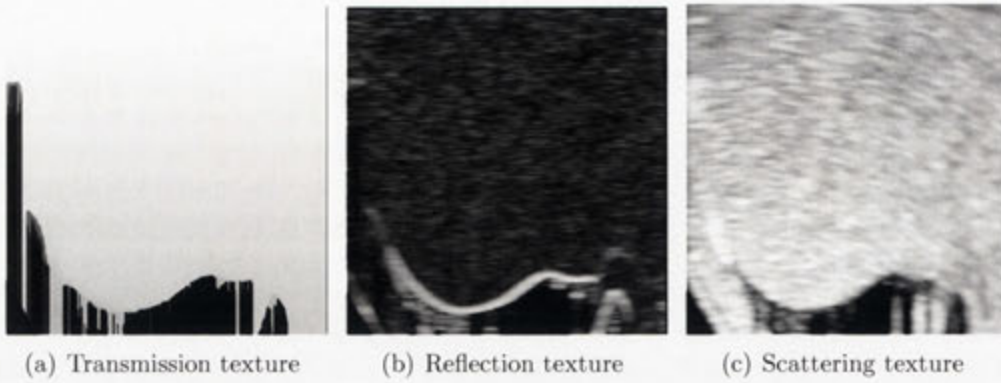


Figure 6.8: Intermediate results from pre-scan conversion stage (a) Transmission scan line image, no filtering applied. (b) reflection scan line image, Hann window applied. (c) Scattering scan line image, Hann window applied.

The ultrasound simulation may require a re-mapping of the CT values so that they can be directly used in equation (6.1). A transfer function lookup texture is used for efficient re-mapping of CT values.

- *Pre-Scan Conversion Stage:* For efficient computation, equation (6.4) can be reformulated as the convolution of the scan line data with an appropriate 1D window function. This computation is performed in the pre-scan conversion stage. Fig. 6.8 shows the content of various textures resulting from pre-scan conversion stage for a sample ultrasound simulation of the liver.
- *Scan Conversion Stage:* This stage is used to convert scan line data into a 2D or 3D Cartesian representation. Scan conversion is implemented by backward warping on the GPU using a specialized fragment shader for each probe geometry and dimension. We use the GPU's built-in bilinear interpolation for maximum performance.
- *Post-Scan Conversion Stage:* 2D and 3D simulated images may have to be filtered for improved visual quality according to (6.6). This requires convolution with the appropriate 2D or 3D filter which is implemented by a fragment shader on the GPU. Separable kernels are used in conjunction with two/three render passes for 2D/3D filtering, where possible, to improve the performance. Fig. 6.9 shows the content of various textures following the post-scan conversion stage for a sample ultrasound simulation of the liver.
- *Compositing Stage:* In the compositing stage, intermediate results from various sources are combined in a fragment shader according to (6.6) and (6.7),

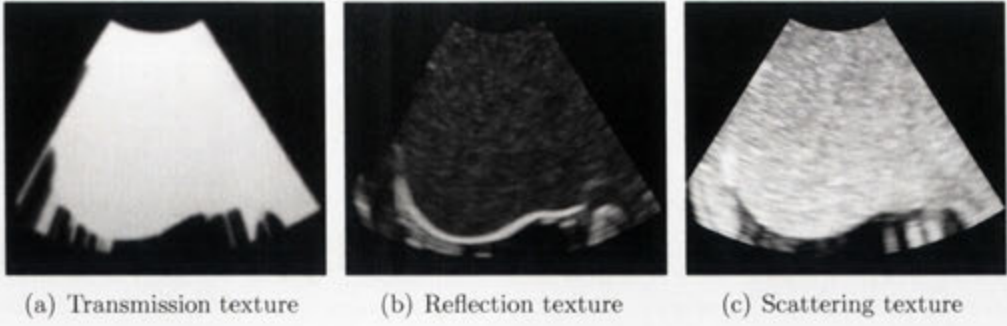


Figure 6.9: Resulting Cartesian images after scan conversion and post-scan conversion stages. (a) Smoothed transmission image (b) Smoothed reflection image (c) Scattering image.

which computes the final value for each pixel and prepares the data for visualization.

6.3.3 Real-Time Visualization

A key component of our real-time visualization is the concurrent display of the simulated ultrasound images within the CT data-set using GPU-accelerated direct volume rendering (see Fig. 6.10). We use the emission-absorption model of light propagation through a translucent volume [115], which is based on the assumption that the volume is filled with light emitting particles and ignores scattering of light. For a ray of light traveling along a direction, parameterized by a variable such as s , the volume rendering integral can be written as

$$I(s) = I(s_0)e^{-\tau(s_0,s)} + \int_{s_0}^s q(\tilde{s})e^{-\tau(\tilde{s},s)}d\tilde{s}, \quad (6.9)$$

with

$$\tau(s_1, s_2) = \int_{s_1}^{s_2} \kappa(s)ds, \quad (6.10)$$

where, $I(\cdot)$ is the intensity of light at a given point along the ray, s_0 denotes the point where the beam enters the volume, and q and τ are emission and absorption coefficients, respectively. The first term in (6.9) describes the background light attenuation by the volume and the second term accounts for the contribution of emitting particles along the ray while taking the distance dependent attenuation of light into account.

In practice, a numerical approximation of the analytical volume rendering integral is used to compute the integral iteratively while traversing the volume, either

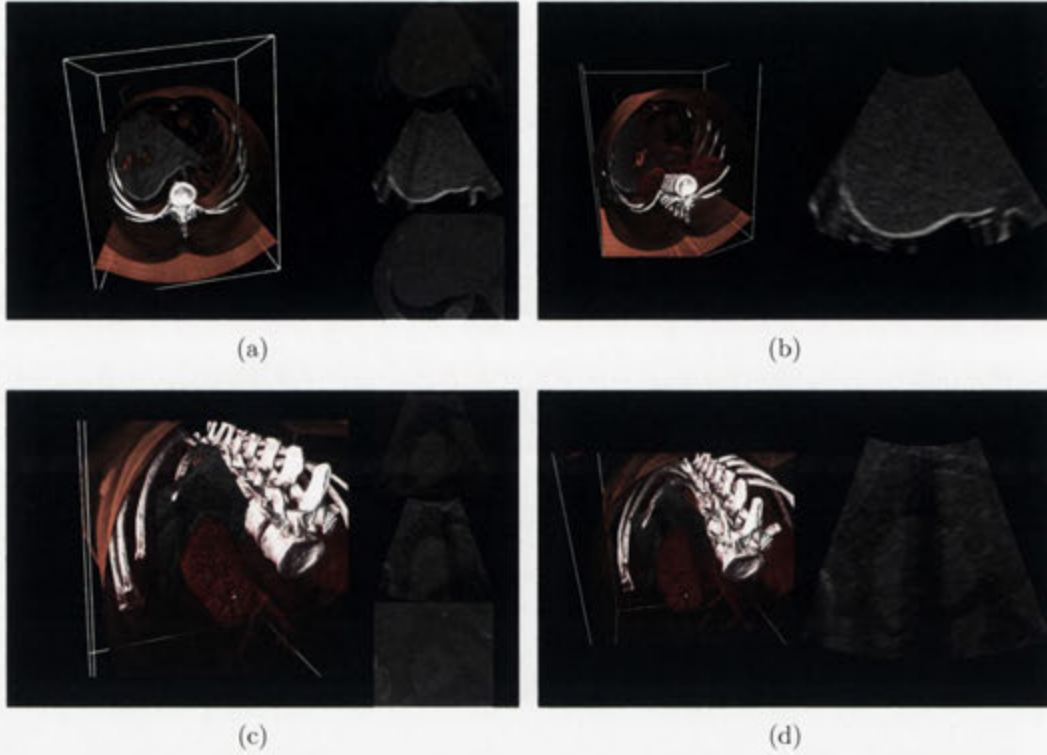


Figure 6.10: Real-time direct visualization of a CT volume and simulated 2D ultrasound. (a,b) Simulation with a wide angle curvilinear transducer scanning the liver - lung boundary, (c,d) Simulation with a narrow angle curvilinear transducer scanning the left kidney. Note the occlusion artifacts due to rays intersecting the ribs.

in a front-to-back or back-to-front fashion along the viewing direction using alpha blending. For a detailed treatment of the subject the reader is referred to [114].

In recent years, GPU-accelerated ray casting has emerged as the de facto standard for high quality real-time direct volume rendering [114, 116–118]. The algorithm owes its popularity to its easy and straightforward implementation on modern GPUs compared to other volume rendering techniques such as texture slicing [114]. Furthermore, the algorithm lends itself to optimization well and is highly adaptable for various visualization tasks.

Despite an exponential improvement in computing capability of GPUs in recent years, volume rendering of 3D medical images remains a computationally expensive task. Various techniques need to be employed in order to achieve real-time high quality rendering. We briefly describe the methods, we employed for achieving interactive frame rates.

- *Deferred rendering:* Our renderer is implemented by a multistage rendering pipeline. To improve the overall performance, each stage of the pipeline is updated only if parameters affecting the stage itself are changed or any

previous stage is updated. Costly operations in the shaders are deferred to the latest possible phase or avoided completely if their contribution is negligible for the final image.

- *Early termination:* We use front-to-back compositing along the viewing ray when computing the volume rendering integral which allows us to terminate the computations if the accumulated opacity is saturated.
- *Volume culling:* With a typical transfer function, a large number of voxels in a volume are fully transparent, and thus do not contribute to the finally rendered image. To improve the performance, no computations should be performed for these voxels. We employ an octree³ to partition the voxels of the volume into cells. Each cell is initialized with the intensity range of the enclosed voxels. For a given transfer function non-contributing cells are culled when computing the ray entry and exit positions. Thus, the ray casting algorithm only renders the volume enclosed by the active octree cells and skips the invisible areas in front and behind the visible volume.
- *Sampling frequency:* The sampling frequency along a ray, is the basic parameter that provides a trade-off between the computational performance versus quality. The quality cannot be improved arbitrarily by increasing the sampling frequency. As such, we use the lowest sampling frequency, beyond which quality improvement is not noticeable. One way to achieve a higher quality at a lower sampling rate is to avoid using a regular sampling grid. A low sampling rate with equidistance samples exhibits visually displeasing grid artifacts also known as rings. Using a random offset, on an otherwise equidistance sampling, removes this artifact and allows the sampling rate to be reduced, without loss of quality, in the interest of improved performance.
- *Classification:* The appearance and visibility of voxels is computed in the classification stage of the volume rendering algorithm. Classification is implemented by texture lookups in transfer function lookup textures. Traditional 1D transfer function tables require a high sampling frequency of the volume data due to high frequencies introduced by the transfer function, e.g. in semi-transparent rendering of tissue interfaces. To deal with this problem and to reduce the volume sampling frequency pre-integrated [119] and post-color attenuated [120] classification techniques have been introduced. Both

³An octree is a tree data structure commonly arising from partitioning a 3D space by a recursive subdivision into eight octants. Hence, each node in an octree has up to eight children.

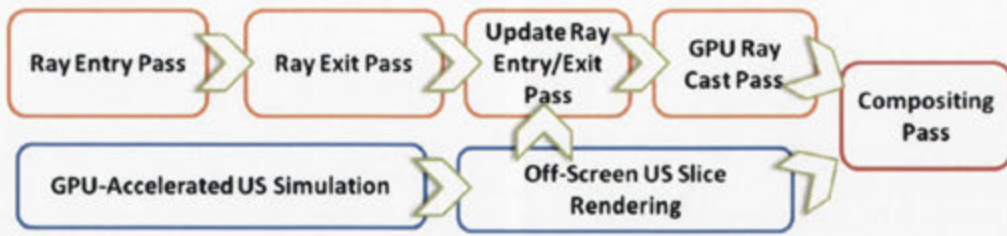


Figure 6.11: Visualization pipeline

approaches pre-compute the volume integral between each two sample values and thus allow a reduction of the sampling frequency while maintaining a high quality. Pre-integrated transfer functions give the best visual results, however the lookup table update is computationally expensive, thus for interactive classification we use post-color attenuated transfer functions.

6.3.4 Visual Consistency of Ultrasound Rendering

The simulated ultrasound is rendered as an opaque plane/volume within the CT DVR space. Missing or incorrect interaction of an opaque geometry, e.g. ultrasound image plane, with a 3D volume disturbs the visual perception of the anatomy and its depth. To correctly integrate the simulated ultrasound within the volume, we adjust the ray start and stop positions in the corresponding textures prior to the ray casting pass. For every valid ray start position a fragment shader checks whether it is occluded by the ultrasound plane by comparing the ray start and the ultrasound plane depth values. If the ultrasound plane is closer to the viewer than the ray start position then it is cleared with zero, so that no ray is cast. For the ray stop positions a similar test is performed. If the ultrasound plane depth value is smaller than the ray stop position, the depth value is back-projected to the normalized volume coordinates and replaces the ray stop position. Thereby, rays are correctly stopped at the first opaque geometry surface in the viewing direction. This is demonstrated in Fig. 6.12(b).

6.3.5 Visualization Pipeline

The described techniques are utilized in the rendering pipeline for displaying the simulated ultrasound images and the volume rendered CT image. Fig. 6.11 and Fig. 6.12 depict various stages of the rendering pipeline and a sample visualization of the pipeline, respectively. In the ray entry and ray exit passes (Fig. 6.12(a), Fig. 6.12(b)), the front and back faces of the volume's bounding geometry are ren-

dered into two textures that store the ray start and stop positions in the normalized volume coordinates at each texel. In the next pass, the ultrasound field of view is rendered in the same 3D space as the volume. The ultrasound field of view is texture mapped with the result of the ultrasound simulation (Fig. 6.12(c), Fig. 6.12(d)). The depth image from this pass is used to update the ray start and stop positions stored in the corresponding textures prior to the ray casting pass (Fig. 6.12(e)). The rendering results of the ray casting pass and the ultrasound plane are composited in the final rendering pass using alpha blending to yield the final image (see Fig. 6.12(f)).

6.3.6 User Interface

We have developed an application for the GPU-accelerated ultrasound simulation to display the simulated ultrasound images in 2D and 3D using different visualization techniques in real-time. A screen-shot of the application's user interface is shown in Fig. 6.13. The user interface consists of four main views and two widgets for adjusting the simulation parameters and the direct volume rendering transfer function:

- *3D View:* The 3D view displays 3D CT volume and the ultrasound plane within the 3D volume. The 3D image is rendered using standard volume rendering techniques and the ultrasound image is texture-mapped onto the corresponding plane within the 3D volume. The user can change the details of the volume rendering (e.g. display internal organs, vasculature, bones or skin surface) in real-time by changing transfer function parameters.
- *Ultrasound View:* displays the simulated 2D/3D ultrasound image. For 3D ultrasound, the user can choose between the 2D display of coronal, sagittal or axial multi-planar reconstructions (MPRs) or a 3D direct volume rendering of the simulated ultrasound volume (see Fig. 6.20, 6.21 and 6.22).
- *CT View:* The CT view displays an MPR of a CT plane that corresponds with the current position, orientation and field of view of the ultrasound image.
- *Combined View:* shows the fusion of the ultrasound and CT images and allows the user to easily compare ultrasound and CT features.

Ultrasound simulation and visualization parameters can be adjusted interactively (see Fig. 6.13 (a), the depicted simulation parameters are for the model by

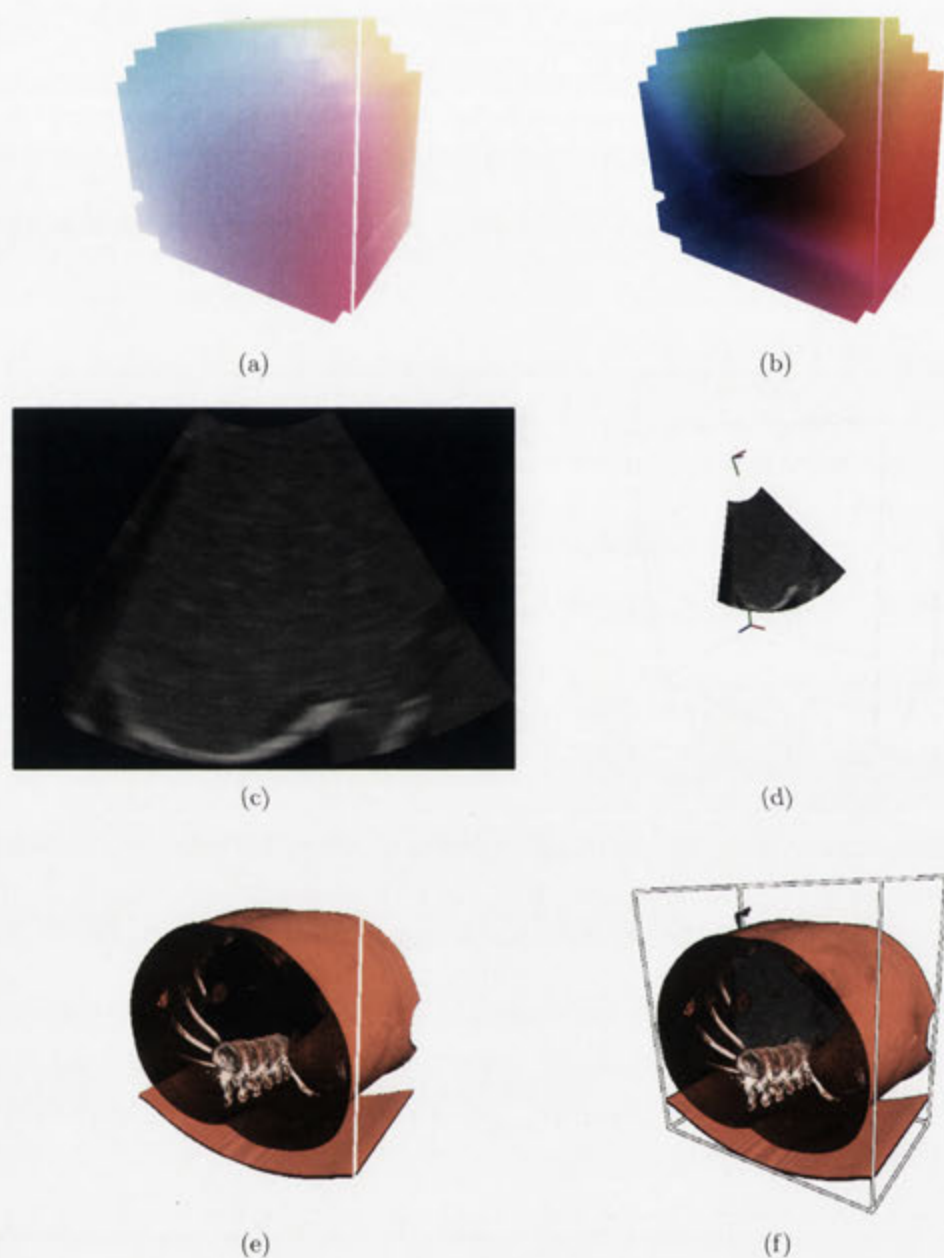


Figure 6.12: Different stages of the visualization pipeline. (a) Ray Start and (b) Ray end coordinates stored in RGB textures. Note ray stop positions for pixels on the ultrasound image plane. (c) Simulated 2D ultrasound image (d) Rendering of the texture-mapped ultrasound image in 3D. (e) Direct volume rendering of the CT image. (f) Composition of CT DVR and simulated ultrasound.

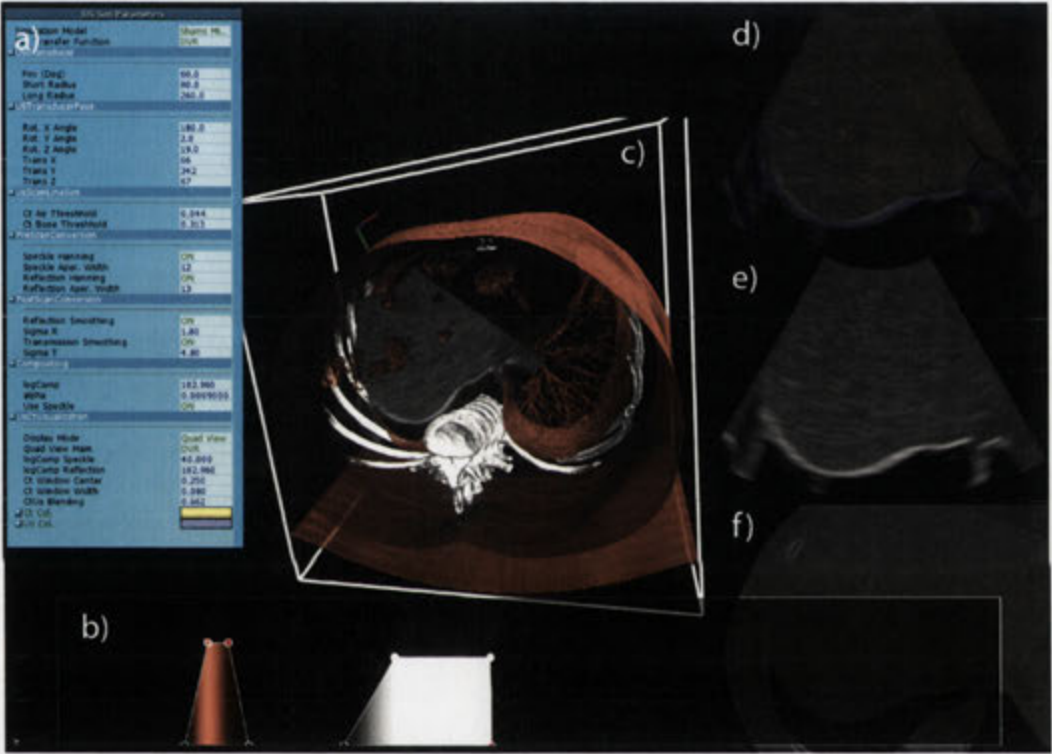


Figure 6.13: Screen-shot of the application: a) application options, b) transfer function widget, c) 3D view, depicting shaded DVR of the CT data and the 2D ultrasound image, d) blending of the simulated 2D ultrasound image and corresponding CT MPR, e) simulated 2D ultrasound image, and f) CT MPR corresponding to the ultrasound image plane.

Shams et al. [104]). The parameters are organized in groups. Certain groups are shared among all simulation models, others are specific to a particular ultrasound simulation model.

- *Transducer Geometry and Pose*: allows for the selection of the probe geometry (i.e. linear or curvilinear), setting the probe position and orientation, field-of-view, and minimum and maximum penetration depth.
- *Scan Line Traversal*: parameters in this group affect the scan line traversal stage. For instance, for the model in [104] these are the air and bone segmentation thresholds.
- *Pre/Post-Scan Conversion*: the options include type of filters, window sizes, standard deviation of the filters, etc.
- *Compositing*: the options include log compression, blending factors and Boolean flags denoting whether certain operations should be executed in the compositing shaders.
- *Visualization*: the options include CT window level and window width for 2D CT slice visualization, and blending factors and colors for the combined CT/ultrasound visualization.

6.3.7 Computational Performance

The ray-based simulation of ultrasound is very efficient on the GPU. In this section, we present detailed performance results for the two main application areas of the simulation framework, simulation and visualization in ultrasound training and simulation for registration of ultrasound and CT images. The requirements for the two applications are different. Ultrasound training requires more realistic simulation of ultrasound images and uses a more accurate simulation model, whereas for registration only a few ultrasound specific effects have to be simulated and a more simplified ultrasound model can be employed. We first describe the test environment and the data-sets and parameters used for the performance evaluation. Then, we describe the performance of the simulation and simultaneous visualization for ultrasound training using the ultrasound model presented earlier. We conclude with an analysis of the throughput performance of the ultrasound simulation for registration of ultrasound and CT images using the ultrasound model by Wein et al. [103].

The performance of the ultrasound simulation and visualization was evaluated on three computers:

1. AMD Opteron 165 CPU (2x 1.8 GHz), 2 GB RAM, AMD/ATI Radeon 1950Pro with 256 MB RAM (Shader Model 3.0), Win. XP (32-bit)
2. Intel Core Duo 2 CPU (2x 2.66 GHz, mobile), 4 GB RAM, NVIDIA Quadro FX3600M with 512 MB RAM (Shader Model 4.0), Win. Vista (64-bit)
3. Intel Core Duo 2 CPU (2x 2.66G GHz), 4 GB RAM, NVIDIA Quadro FX5600 with 1.5 GB RAM (Shader Model 4.0), Win. Vista (64-bit)

For our performance measurements, we used a CT volume of the abdomen of a human subject with a resolution of $512 \times 512 \times 484$ voxels (16-bit, 242 MB). A speckle volume of the same size was pre-computed from the CT data (32-bit float, 484 MB). The full-size volumes were used with Quadro FX5600 (1.5 GB RAM) but the volumes were down-sampled for Quadro FX3600M and Radeon 1950Pro cards to fit within the GPU's memory.

6.3.8 Performance of Simulation and Visualization in Training Applications

Training applications require interactive frame rates for operation of a virtual transducer and provision of a smooth and uninterrupted visual feedback. In this section, we first evaluate the combined performance of the simulation and visualization and then compare the performance of simulations on the different GPUs and with a CPU implementation.

Visualization is typically the more time-consuming part of the algorithm. The performance of visualization is dependent on many parameters, e.g. direct volume rendering technique, local illumination, and the chosen transfer function. For our experiments, we adjusted the parameters for high quality visualization using pre-integrated classification for DVR, local illumination with Blinn-Phong shading and on the fly-gradient evaluation. A resolution of 640×480 pixels was used for rendering, the number of samples per ray was set to 512, early ray termination and empty space leaping optimizations were also activated.

Table 6.1 shows the average frame rate of the combined simulation and visualization as the operator varies simulation and visualization parameters. Changes in relative orientation or position of the volume with respect to the camera, the ultrasound transducer with respect to the volume, or transducer geometry affect the performance. This is due to the fact that these changes require the entire rendering

pipeline to be re-executed. As can be seen, the algorithm performs interactively on higher end GPUs such as FX5600, under all conditions. The performance is equally good for mainstream GPU models (e.g. 8800GTX/GTS and 9800GTX) that have around the same number of stream processors as FX5600. However, for lower end GPUs the rendering quality has to be reduced in order to achieve interactive frame rates, under all conditions.

Param. Change	Radeon 1950Pro	Quadro FX3600M	Quadro FX5600
None - Sim. Only	45	79	162
Volume Pose	5	15	32
Transducer Pose	4	16	35
Transducer Shape	4	15	34
Sim. Param.	40	76	157
Transfer Function	9	31	47

Table 6.1: Performance in frames per second for the combined simulation and visualization.

The performance of the simulations were also measured by throughput in megapixels rendered per second for varying numbers of scan lines and samples, and ultrasound image resolutions (see Table 6.2 for benchmark configuration details). The results were compared with the throughput of a CPU implementation measured on an Intel Core 2, Quad 3.0 GHz processor. The results are given in Fig. 6.14. Unlike the GPU version, the throughput for the CPU implementation does not (noticeably) vary with the image size. The GPU implementation outperforms the CPU by up to ~ 20 times (see Fig. 6.15).

Benchmark Index	Scan lines	Depth Samples	Ultrasound Image Resolution
1	256	256	256×256
2	512	512	512×512
3	512	512	640×480
4	512	512	800×600
5	1024	1024	1024×1024

Table 6.2: Benchmark configuration parameters for performance evaluation of single 2D ultrasound image simulation.

6.3.9 Performance of Simulation for Registration Applications

Registration of ultrasound and CT images requires the repeated simulation of ultrasound images at various orientations and positions from the CT data during

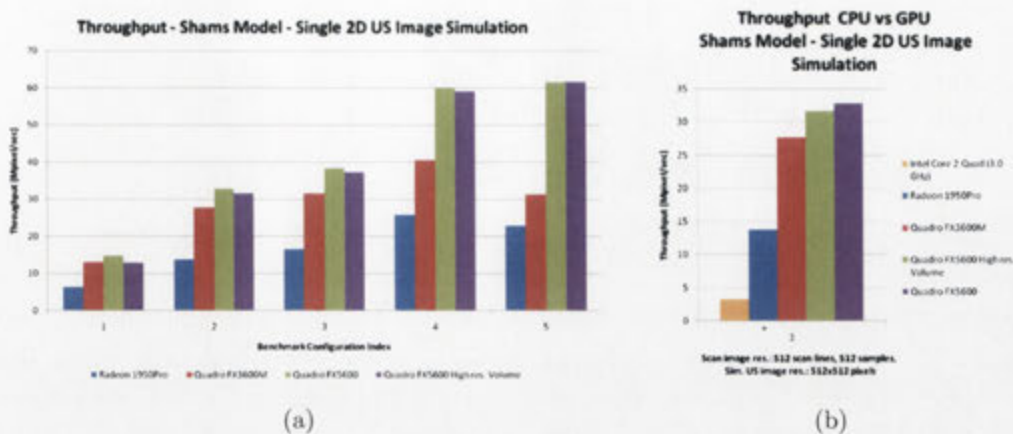


Figure 6.14: (a) Benchmark results for simulation of a single 2D ultrasound image using the simulation model by Shams et al. for benchmark configurations denoted in Table 6.2. (b) CPU versus GPU performance comparison for simulation of a single 2D ultrasound image: 512 scan lines with 512 samples, 512 × 512 pixels image resolution.

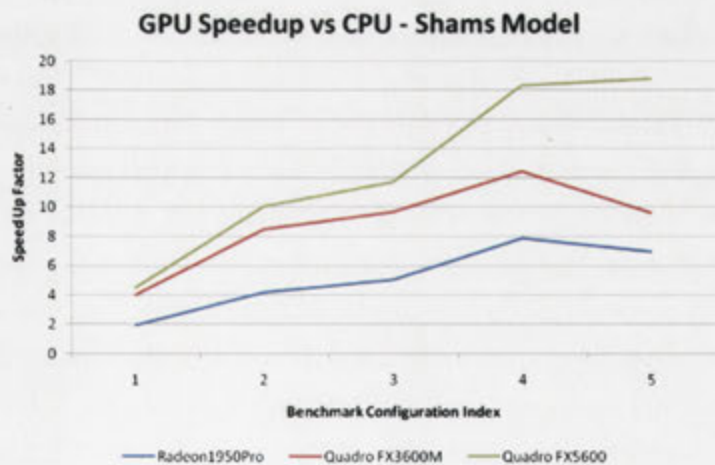


Figure 6.15: GPU speedups for the simulation of a single 2D ultrasound image using the model by Shams et al. for benchmark configurations in Table 6.2.

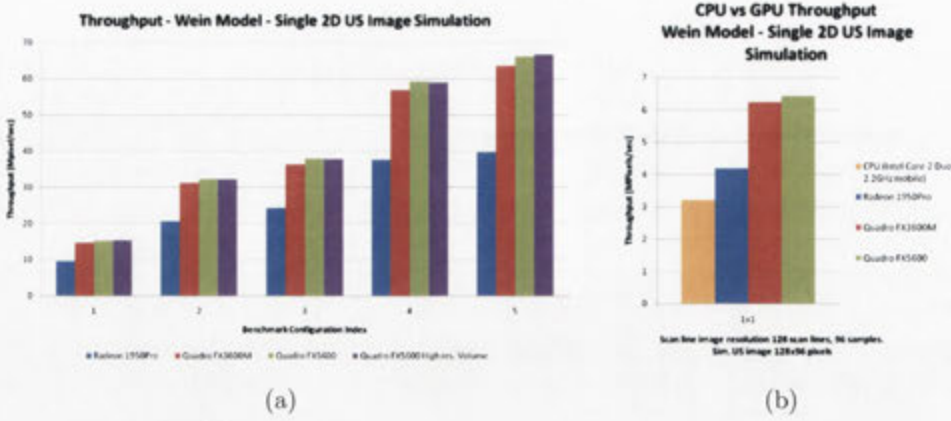


Figure 6.16: (a) Simulation performance for a single 2D ultrasound image using the benchmark configuration parameters specified in Table 6.2. (b) Comparison of GPU and CPU throughput for simulating a single 2D ultrasound image of 128×96 pixels.

optimization of the registration parameters. The simulation of a single 2D ultrasound image using the model in [103] barely utilizes the computational resources of the GPU. Fig. 6.16 depicts the throughput for a single 2D ultrasound image using the scan line and ultrasound image resolutions given in Table 6.2. The throughput is limited by the number of active fragment shaders/stream processors and control program overhead in the scan line traversal stage. As can be seen in Fig. 6.16(a), the throughput improves as the size of the ultrasound image increases, since GPU resources are being more optimally utilized for larger images. The authors of [103] kindly provided us with the timings of their CPU ultrasound simulation C++ implementation on a 2.2 GHz Intel Core 2 Duo mobile processor. The simulation of a single 2D ultrasound image, 128×96 took ~ 3.5 ms. To compare the performance, we used this value for estimation of the GPU speedups compared to the CPU for the simulation of single 2D ultrasound images (see Fig. 6.16(b)) and multiple 2D ultrasound images on the GPU (see Fig. 6.19) of the same resolution.

The key to increase the throughput of the simulation is to process more fragments in a single pass of the scan line simulation stage. We achieve this by packing multiple ultrasound images into tiles of a large texture on the GPU. In each simulation pass, multiple ultrasound images are processed resulting in an improved GPU hardware utilization and increased data throughput per second.

Fig. 6.17 and Fig. 6.18 depict the throughput achieved by the parallel simulation of multiple ultrasound images. The throughput increases by the number of image tiles. Using a tile configuration of 32×16 images, each with a 256 scan lines with 256 samples and an ultrasound image resolution of 256×256 pixels, we achieved a

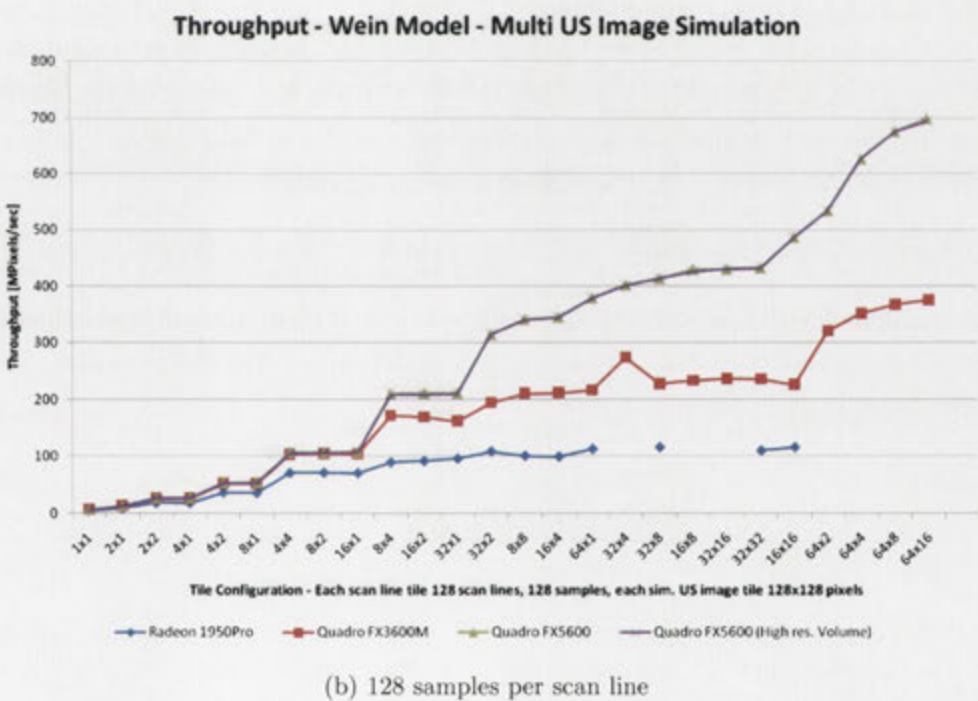
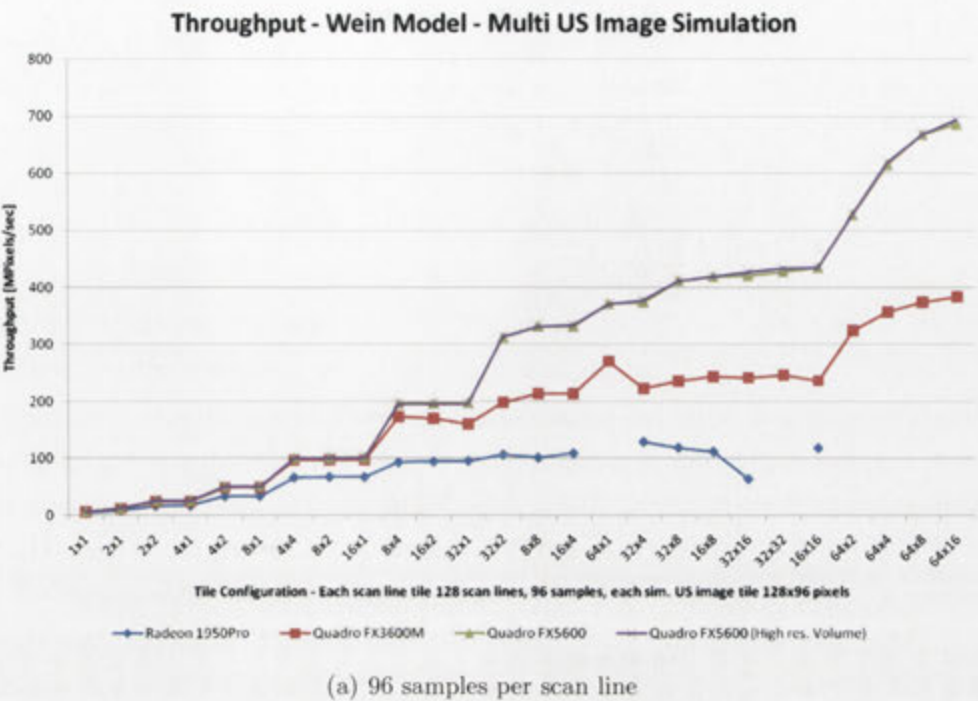


Figure 6.17: Throughput [MPixels/sec] for different image tile configurations: the throughput increases with the number of tiles and is typically optimal for square configurations.

throughput of >700 MPixels/sec on an NVIDIA Quadro FX5600 board.

Fig. 6.19 depicts the speedup for our multi-image GPU simulation compared to the CPU implementation by Wein et al. [103] for the simulation of ultrasound images of 128×96 pixels resolution and 128 scan lines with 96 samples. With a Quadro FX5600 a speedup of more than 200 times can be achieved.

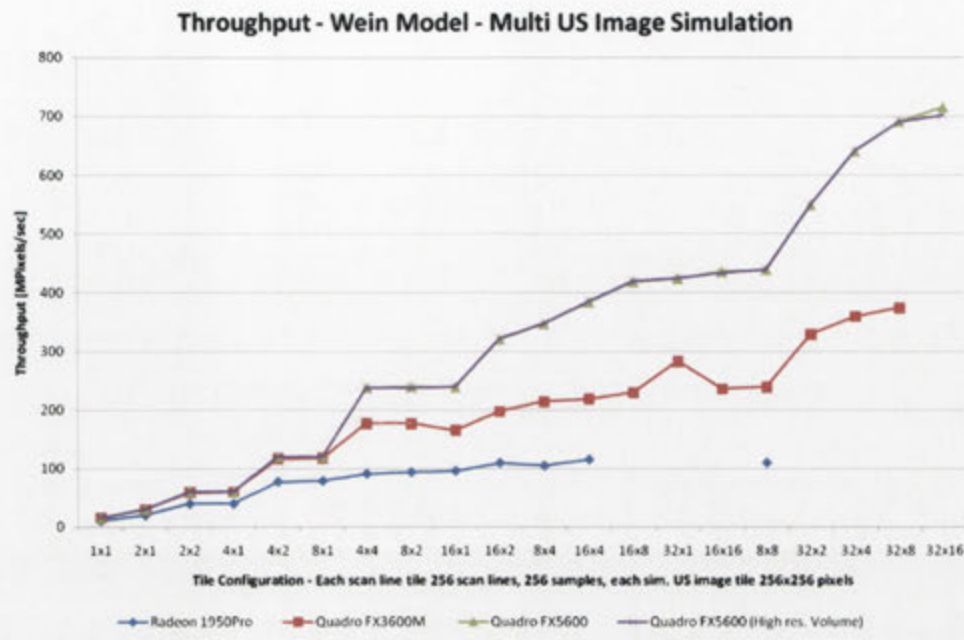
6.4 Discussion

We are investigating further enhancement of our ultrasound model including absorption and refraction based on labeling tissue types, multiple echoes, and simulating tissue deformation due to pressure by the ultrasound probe.

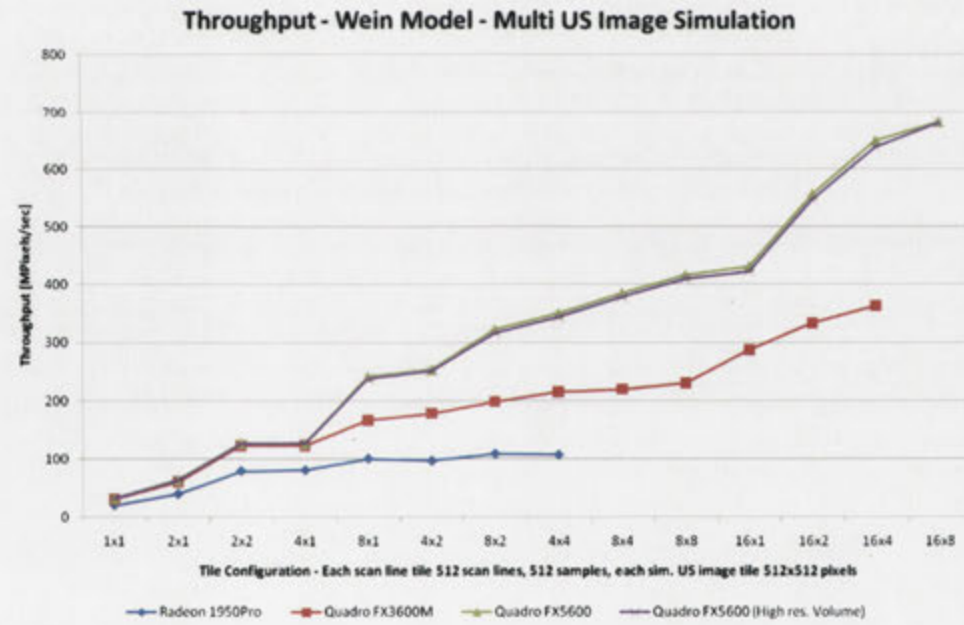
One limitation of the method is that it may not be readily used for prenatal training due to dependence on CT images as input. Prenatal CT scans are rarely performed and are typically reserved for cases with complications or when the fetus is deemed unviable. Use of MRI or synthetic models can be investigated.

Another interesting application of ultrasound simulation is real-time registration of 2D ultrasound to a 3D volume. We are hypothesizing that a more accurate simulation of the ultrasound, facilitates registration of an actual ultrasound against a CT volume. This is driven by the intuition that comparing an actual ultrasound with a closely simulated one, reduces the burden on the design of the similarity measure and the optimization algorithm and can potentially lead to interactive registration. This application is currently being investigated.

Simulating the scattering image is time-consuming and requires a cluster of CPUs to be practical. This is less of a burden as 4- and 8-core systems are becoming commonplace. GPU implementation of the scattering simulation is worthwhile as GPUs with up to 240 cores are available at retail prices and the processing can be easily distributed.



(a) 256 samples per scan line



(b) 512 samples per scan line

Figure 6.18: Throughput [MPixels/sec] for different image tile configurations: the throughput increases with the number of tiles and is typically optimal for square configurations.

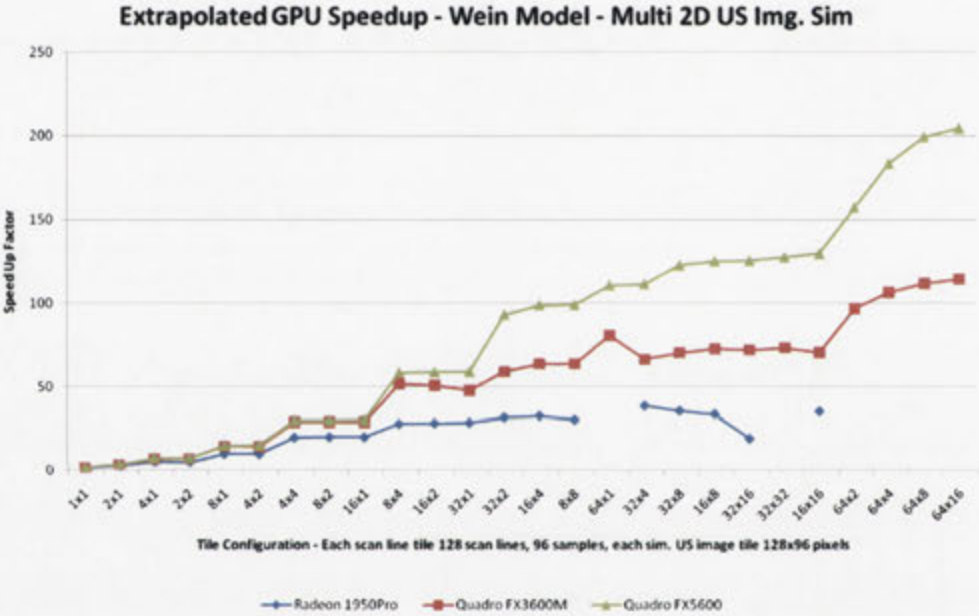
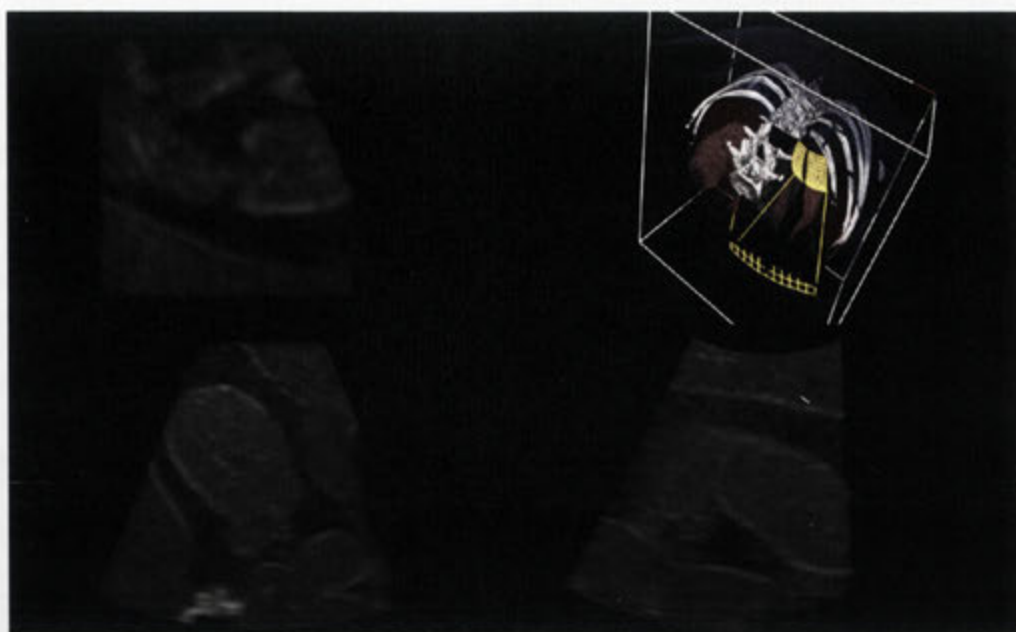
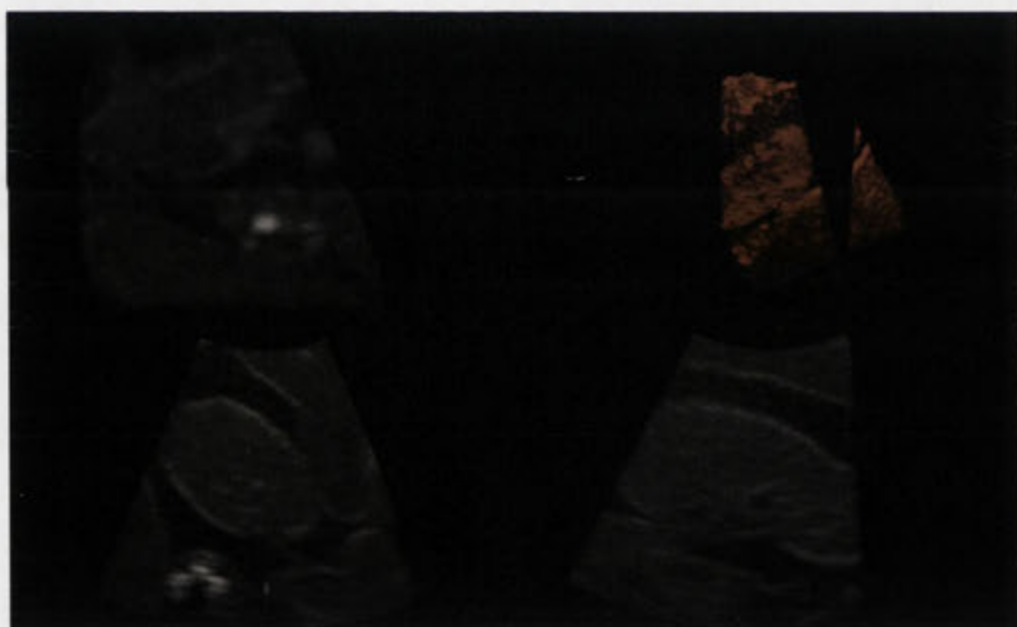


Figure 6.19: Speedups for GPU implementation of Wein’s model and simulation of multiple images of resolution 128×96 compared to performance of CPU implementation for simulation of the same total number of images. CPU implementation performance values estimated from timings provided by Wein et al. [103].

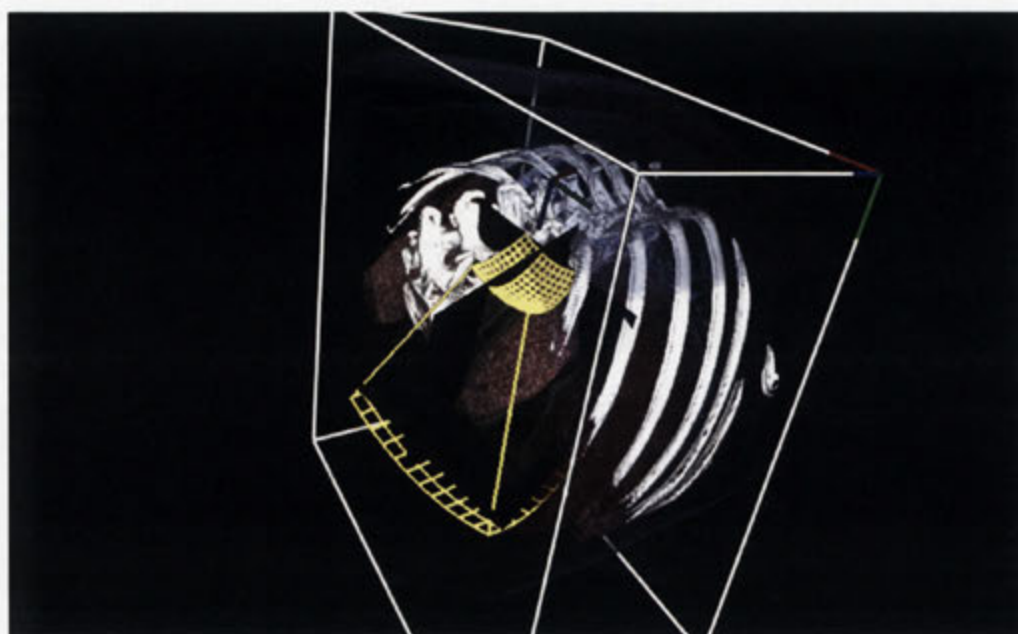


(a)



(b)

Figure 6.20: Screen shots from visualization of simulated 3D ultrasound volume. (a) Coronal, sagittal and axial MPRs extracted from simulated ultrasound volume. Upper-right quadrant: Volume rendering of CT data with wire frame rendering of ultrasound field of view and ultrasound MPRs inside the CT volume. (b) Coronal, sagittal and axial MPRs extracted from simulated ultrasound volume. Upper-right quadrant: Volume rendering of simulated ultrasound volume with texture mapped MPR planes.



(a)



(b)

Figure 6.21: Screen shots from visualization of simulated 3D ultrasound volume. (a) Volume rendering of CT data with wire frame rendering of ultrasound field of view and ultrasound MPRs inside the CT volume. (b) Volume rendering of simulated ultrasound volume with texture mapped MPR planes.

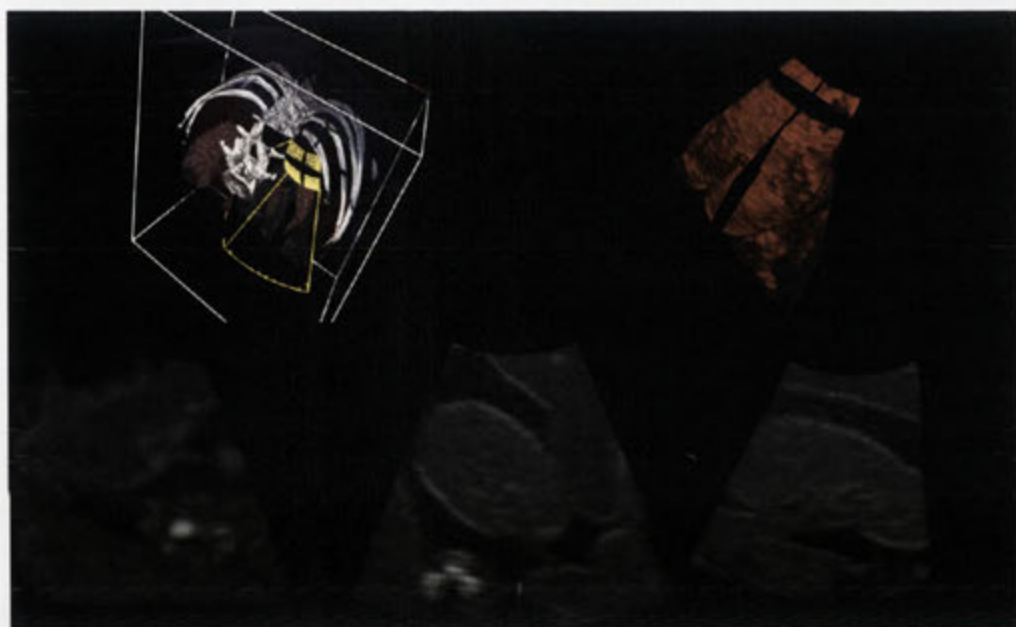


Figure 6.22: Screen shot from visualization of simulated 3D ultrasound volume. Top row, from left to right: Volume rendering of CT data with wire frame rendering of ultrasound field of view and ultrasound MPRs inside the CT volume. Volume rendering of simulated ultrasound volume with texture mapped MPR planes. Bottom row, from left to right: Axial, sagittal and coronal MPR planes extracted from simulated ultrasound volume.

Chapter 7

Conclusions

The traditional approach to solving computational problems, as we have noted in more than one occasion in this thesis, is being challenged now more than ever by massively parallel architectures. Massively parallel systems are no longer limited to super-computing facilities and expensive data centers. A typical desktop system is now a heterogeneous computational powerhouse. One whose potential craves to be unlocked by a fresh approach to software design from conception of algorithms to their implementation. There is enormous potential in clustering these systems to form highly powerful and low-cost clusters to tackle computationally challenging problems.

The advent of this new generation of low-cost high performance computing platforms presents both numerous opportunities and challenges. A plethora of fundamental research questions related to high performance computing on many-core heterogeneous systems needs to be investigated. The overhaul of software platforms, algorithmic design, programming models, and tools required to fully embrace these new technologies is indeed a grand challenge. One notable problem is efficient and practical scalability of applications in clusters of heterogeneous computers, while also ensuring that the associated programs fully utilize all the CPU and accelerator resources.

In the field of medical image analysis, in particular, small clusters of heterogeneous GPUs and CPUs are of practical interest. They can be deployed in radiology departments and operating rooms to solve image analysis and image-based navigation problems at a low cost, with minimal power consumption and within a small footprint. The increased computational resources can be used in a number of ways: (a) to improve the robustness of existing processes through combining the results from different solutions to a problem by consensus, (b) to solve problems that would otherwise be computationally intractable, (c) to enable existing algorithms

to be run faster, even in real-time, to allow adaptation of a range of preoperative tools to intraoperative setups, and (d) to improve productivity through enhanced efficiency of existing tools.

Achieving these goals require a coordinated effort by the medical imaging community to redesign a large set of existing and fundamental algorithms in areas such as registration, segmentation and modeling specifically for massively parallel architectures. A feat which many not be possible without a more enthusiastic adoption of high performance computing within the field. We envisage that existing efforts in this area will become more focused in the next few years. This may pave the way for the organic growth of a *computational medical imaging* community in much the same way that computational physics, chemistry and astronomy have grown out of their respective disciplines.

Appendix A

Rotation Matrix

There are two possible scenarios when one talks about a rotation:

1. *World axes rotation*: rotation of the world coordinates
2. *Object axes rotation*: rotation of an object w.r.t. fixed world coordinates

Rotation of an object w.r.t. to fixed coordinates can be described by the rotation of the coordinates by the same magnitude and in the opposite direction. R_w and R_o in (A.1) describe world axes and object axes rotations by counterclockwise angle θ in \mathbb{R}^2 .

$$R_w(\theta) = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}, \quad R_o(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}. \quad (\text{A.1})$$

According to *Euler's rotation theorem*, a rotation in \mathbb{R}^3 may be described by only three parameters. These parameters are known as *Euler angles*. Rotations in \mathbb{R}^3 can be described by three rotations around the three principal axes or by three rotations around two principal axes where no two consecutive rotations are around the same axis. (A.2) gives world axes rotation matrices around the three principal axes by counterclockwise¹ angles α , β and γ around x , y and z axes, respectively.

¹Counterclockwise/clockwise direction is determined by an observer whose feet are at the origin and is standing in the direction of the axis.

$$\begin{aligned}
R_x(\alpha) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & \sin \alpha \\ 0 & -\sin \alpha & \cos \alpha \end{bmatrix}, \\
R_y(\beta) &= \begin{bmatrix} \cos \beta & 0 & -\sin \beta \\ 0 & 1 & 0 \\ \sin \beta & 0 & \cos \beta \end{bmatrix}, \\
R_z(\gamma) &= \begin{bmatrix} \cos \gamma & \sin \gamma & 0 \\ -\sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}, \\
-\pi < \alpha \leq \pi, \quad -\frac{\pi}{2} < \beta \leq \frac{\pi}{2}, \quad -\pi < \gamma \leq \pi.
\end{aligned} \tag{A.2}$$

For a right-handed coordinate system, there are 24 ways of parameterizing a rotation². When using Euler angles the order in which rotations are applied and the order in which the angles are given must be specified. We specify the parametrization convention by three letter axes designations. For example, xyz convention is a rotation around the x -axis followed by rotations around y and z axes.

$$\begin{aligned}
R_{xyz}(\alpha, \beta, \gamma) &= R_z(\gamma)R_y(\beta)R_x(\alpha) = \\
&\begin{bmatrix} \cos \beta \cos \gamma & \cos \alpha \sin \gamma + \sin \alpha \sin \beta \cos \gamma & \sin \alpha \sin \gamma - \cos \alpha \sin \beta \cos \gamma \\ -\cos \beta \sin \gamma & \cos \alpha \cos \gamma - \sin \alpha \sin \beta \sin \gamma & \sin \alpha \cos \gamma + \cos \alpha \sin \beta \sin \gamma \\ \sin \beta & -\sin \alpha \cos \beta & \cos \alpha \cos \beta \end{bmatrix}.
\end{aligned} \tag{A.3}$$

In this document, rotation implicitly refers to world axes rotation. To avoid confusion, we directly work with rotation matrices as opposed to parametrization by Euler angles, where possible. Any parametrization by Euler angles is done according to the xyz convention.

²12 for world axes and 12 for object axes rotations.

Appendix B

Medical Imaging Coordinate Systems

In this section, we explain the conventions and the terminology used throughout this thesis to specify the manner in which 3D image data is stored and represented to the user.

B.1 Anatomical Directions and Planes

Fig. B.1 depicts the 6 principal directions w.r.t. to a human subject's anatomy: left/right, anterior/posterior, and superior/inferior or for short L/R, A/P, and S/I. Note that use of terms such as back/front and up/down is ambiguous when dealing with different orientations of a subject (e.g. standing up, lying down, etc.) and should be avoided. In the context of human subjects, one may use head/foot term instead of superior/inferior but we prefer the later due to its generality. Fig. B.1 also shows the three principal anatomical planes: *axial* (or *transverse*), *sagittal* and *coronal* planes. These planes are defined as being orthogonal to S/I, L/R, and A/P directions, respectively.

B.2 Voxel Ordering

3D images are stored as streams of intensity values. The order in which these voxels are stored is arbitrary and determined by the scanner. In order to visualize and manipulate the data correctly, one needs to know the voxel ordering. The order of voxels is specified with a three-letter direction code, starting with the fastest changing index to the slowest w.r.t. the subject's orientation. This allows for a volumetric image to be stored in 48 different ways.

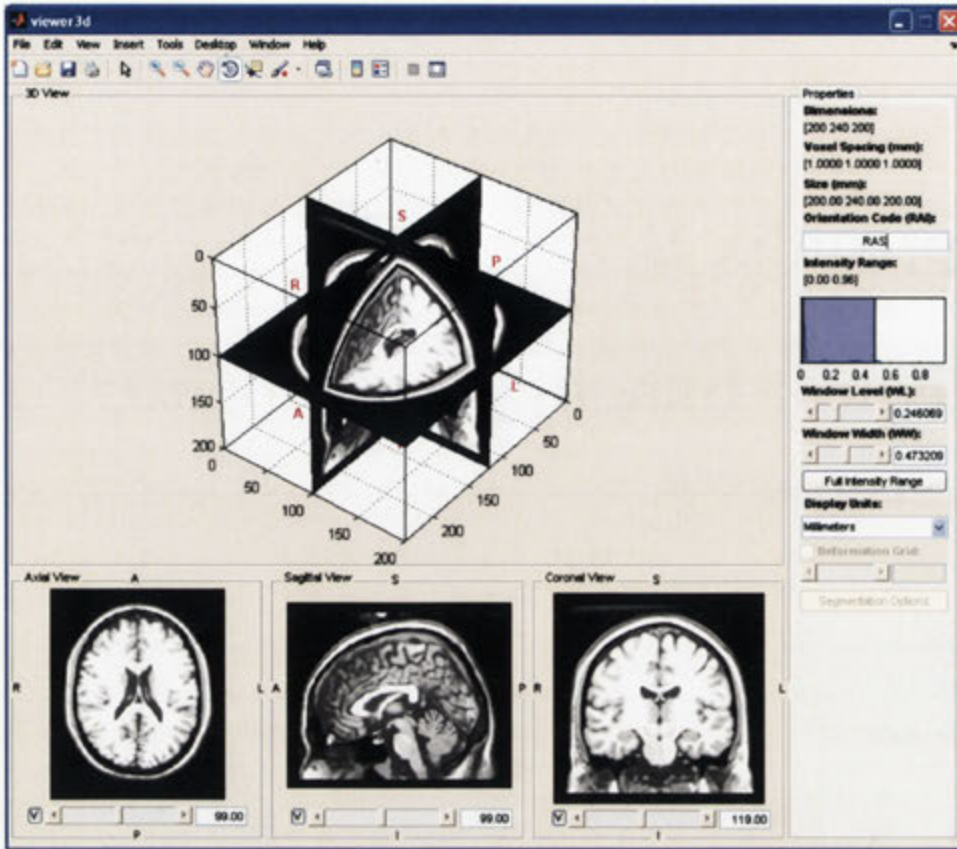


Figure B.1: MATLAB-based implementation of a 3D medical image viewer. The image shows principal anatomical directions and planes.

For example, when we say an image is stored in 'RPS' order, it means that voxels are stored from right to left, in rows that are from posterior to anterior and in slices that are from superior to inferior of the subject. This is called the *from order code* specification which is used by most storage formats as opposed to the *to order code* specification where each letter specifies the direction to the respective anatomical orientation. We use 'from codes' to specify storage orders in this thesis.

B.3 Subject Coordinate System

Medical images typically depict a single subject. Software that visualizes or manipulates the image should use the same coordinate system for all subjects regardless of the voxel ordering of the underlying data for consistency and improved user experience. The origin of the coordinate system is chosen in the center of the volume and the direction of the principal axes x , y and z are specified by a three-letter direction code. It is customary, however confusing, to use *to order codes* to specify

the subject's coordinate system. For example, an 'RAS' coordinate system refers to one where the x -axis points to the right of the subject, y -axis points to the anterior of the subject and z -axis points to the superior of the subject. Unless otherwise specified, we use 'RAS' coordinate system in this thesis.

B.4 2D and 3D Display of Anatomical Planes

A common method for displaying 3D medical images is to show 3 principal anatomical planes in 2D together with a 3D display of the selected principal planes. Fig. B.1 shows our MATLAB-based implementation of a 3D medical viewer. The 3D image is given in an 'RAS' coordinate system and the 2D views depict the so called *radiological* point of view.

Appendix C

Hardware Configuration

The following summarizes the hardware specification of a number of systems that were used in different experiments. The relevant table number is referenced within the main text where appropriate.

Table C.1: Host Specification (CPU)

Processor	AM2 Athlon 64×2 6000+ 3.0 GHz
Memory	4 GB, 800 MHz DDR2
Motherboard	ASUS M2N-SLI Deluxe
Operating System	Windows XP 32-bit

Table C.2: Device Specification (GPU)

Model	GTX 8800	GTX 280	GTX 295
Number of GPUs	1	1	2
Multiprocessors per GPU	16	30	30
Cores per multiprocessor	8	8	8
GDDR3 memory	768 MB	1 GB	1792 MB
Memory interface	384 bits	512 bits	896 bits
Nominal memory bandwidth	86.4 GB/s	141.7 GB/s	223.8 GB/s
Shared memory per block	16 KB	16 KB	16 KB
Registers per multiprocessor	8 KB	16 KB	16 KB
Max threads per block	512	512	512
Max active warps per multiprocessor	24	32	32
Warp size	32	32	32

Appendix D

Intraoperative Ultrasound Probe Calibration in a Sterile Field

D.1 Introduction

Strict sterility requirements impose restrictions on the type of equipment and its handling in an operating room (OR). In this chapter, we propose three methods for intraoperative calibration of a tracked ultrasound probe under sterile conditions. We categorically call these methods *air calibration* to contrast them with the common phantom-based calibration methods that employ a coupling medium (e.g. a water-bath). The methods each consist of a preoperative and an intraoperative calibration stage. The preoperative stage is performed once in a non-sterile environment where any of the existing calibration methods can be used. The intraoperative stage is performed before each intervention in the OR. To minimize impact on the interventional work-flow, we required that the intraoperative calibration took less than 10 minutes, produced a robust result and was easy to perform.

The proposed calibration methods are designed to be used with a navigation system for laparoscopic [121,122] and endoscopic [123] transgastric interventions. The system had been initially validated *in vivo* on porcine models with a non-survival protocol [124] where sterility of the operating room (OR) was not a requirement. In moving from porcine to human subjects, sterility of the experimental equipment became a primary concern. This imposed limitations on the type of equipment and its handling in the sterile field (and during the disinfection process for endoscopy). As we have found, introduction of new material and practices into an OR is complex; it requires extensive iterative development of appropriate protocols and eventual certification. The process can postpone experiments for months. In this chapter, we discuss one specific challenge that we encountered with the cali-

bration of ultrasound and new methods that were devised around these practical limitations.

The aforementioned navigation system consists of a laparoscopic ultrasound (LUS) or an endoscopic ultrasound (EUS) probe. An electromagnetic (EM) sensor (Ascension Technology) is attached close to the transducer on the probe where the relative position of the sensor w.r.t. the ultrasound plane can be maintained. In prior porcine experiments, Polyolefin tubes (shrink wrap) were used to secure the EM sensor. A standard *single-wall phantom* calibration was then performed [125] in a water-bath prior to the operation. The calibration procedure need not be repeated as long as the EM sensor is not removed.

In human subjects, however, there is a concern that biological sediments may not be properly removed by the disinfection or sterilization process, if the sensor is not detached from the instrument. This means that the EM sensor may not be configured permanently, but rather is attached after the components are sterilized separately and supplied in the sterile field of the OR. As such, there is a need for intraoperative calibration of the instrument in the sterile field.

An intraoperative calibration method has been proposed by Chen et al. in [126], who designed a *double-N* phantom that can be disassembled for sterilization and reassembled for the operation. Sub-millimeter accuracy is reported in conjunction with an optical tracking system.

There are good arguments against ultrasound calibration in the OR with phantoms and liquids. Sterilization, phantom construction, and phantom assembly issues aside, accurate calibration is time-consuming and a delicate task. For example, correction for the speed of sound [127] may involve controlling the water temperature or creating water-glycerol or water-ethanol solutions. The more complicated an engineering solution, the less likely it is to be integrated into a clinical work-flow.

A phantom-less method for quality-control of calibration parameters is given by Boctor et al. [128]. The method can recover a sub-set of calibration parameters. In [129], Wein and Khamene propose a method that employs spatial consistency of two orthogonal freehand ultrasound sweeps of a region of interest to determine calibration parameters. The method is suitable for transcutaneous ultrasound probes and cannot be readily applied to the calibration of endoscopic or laparoscopic ultrasound.

D.2 Air Calibration

We propose three calibration methods (*Mold Calibration*, *Funnel Calibration*, and the *Closest Point Calibration*) that do not involve use of a phantom or a coupling medium (such as a water-bath) and can be performed easily by technicians in the OR. The third method has the added advantage that it uses only material already available in the OR and thus does not require an approval process. We categorically call these techniques *air calibration* methods.

These methods partition the process into preoperative and intraoperative calibration steps. Preoperatively, the ultrasound probe can be calibrated using any phantom-based method such as the single-wall phantom, Cambridge phantom, 3-wire phantom [125], or Hopkins phantom [130]. In a conventional calibration setup the following coordinate systems are defined, the tracker (world) coordinates C_T , the sensor coordinates C_S , and the ultrasound image coordinates C_U as shown in Fig. D.2. The conventional calibration problem determines ${}^S T_U$ the transformation matrix from the ultrasound image's coordinate system to the sensor's coordinate system.

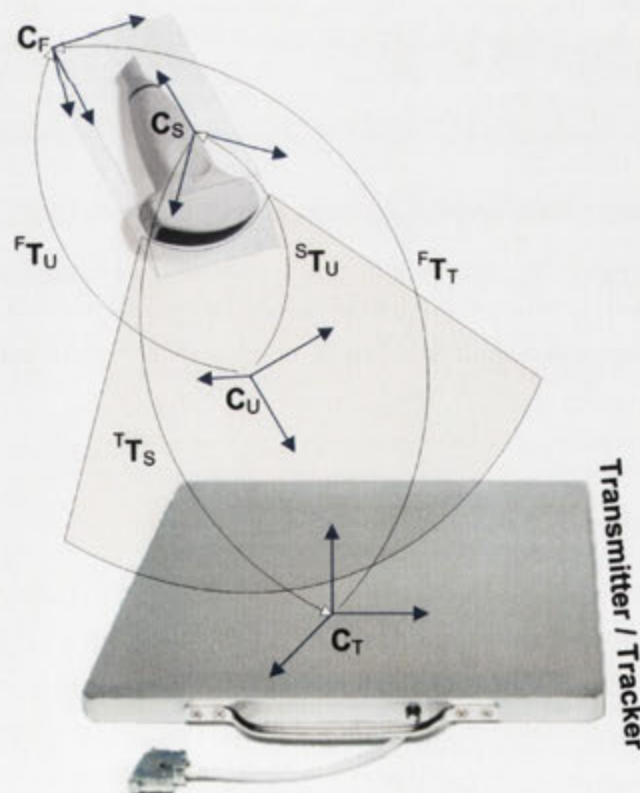


Figure D.1: Coordinate systems and transformations used in air calibration

For air calibration, we introduce an extra coordinate system (C_F) whose position remains fixed w.r.t. the ultrasound transducer. This coordinate system is attached to a physical (mold and funnel calibration) or virtual (the closest point calibration) object that maintains its relative position to the ultrasound plane.

We postulate that we can specify a stable coordinate system that remains fixed w.r.t. to the ultrasound coordinate system. The transformation matrix from the ultrasound plane to this coordinate system is denoted by ${}^F\mathbf{T}_U$. The purpose of the preoperative calibration is to determine this transformation. We first perform a conventional calibration and determine ${}^S\mathbf{T}_U$. Then, the position of C_F is determined w.r.t. the tracker (${}^T\mathbf{T}_F = {}^F\mathbf{T}_T^{-1}$) and we can now determine ${}^F\mathbf{T}_U$ based on known relationships

$${}^F\mathbf{T}_U = {}^F\mathbf{T}_T {}^T\mathbf{T}_S {}^S\mathbf{T}_U. \quad (D.1)$$

During the intraoperative phase, the position of the EM sensor has changed (the sensor has been detached and reinstalled) and ${}^S\mathbf{T}_U$ from the preoperative stage is no longer valid and a new calibration is required. The intraoperative calibration stage consists of measuring the position of C_F w.r.t. the tracker with the same method used in the preoperative step to determine ${}^T\mathbf{T}_F$ and then compute the calibration matrix ${}^S\mathbf{T}_U$ using known matrices

$${}^S\mathbf{T}_U = {}^T\mathbf{T}_S^{-1} {}^F\mathbf{T}_T^{-1} {}^F\mathbf{T}_U. \quad (D.2)$$

So far we have not been specific in setting the fixed coordinate system and its position and orientation in the coordinate frame of the tracking device. In the next sections, we discuss three air calibration methods to achieve this. The first two that require a device to be built are briefly discussed; the third method that uses point-to-surface registration will be explained in detail.

D.2.1 Mold Calibration

The tip of the ultrasound probe is cast to create a mold. We built a mold that fits the inflexible tip of the ultrasound probe. The asymmetries in the probe's shape ensure that the probe fits in the mold in a unique position. A second sensor is attached to the exterior of the mold. During the preoperative calibration stage, the ultrasound plane is calibrated relative to the mold sensor. The position of the mold sensor defines our fixed coordinate system, as the sensor does not have to be removed for sterilization (it does not directly touch the probe). During the

intraoperative phase, a sensor is also attached to the probe and the probe is inserted in the mold. Probe calibration matrix is then computed using (D.2).

D.2.2 Funnel Calibration

This method is a variation of the mold calibration process, in which the mold is built with a sharp pointed end, like a funnel. Preoperatively, the ultrasound plane is calibrated against the tip of the funnel in a manner similar to calibration of a stylus. Since the position of the funnel tip does not change w.r.t. the ultrasound plane, the same process can be repeated intraoperatively to recover the calibration matrix. The funnel calibration requires a single sensor.

D.2.3 Closest Point Calibration

The previous methods require purpose-built appliances. Even when the appliances are built using approved material and are sterilizable, they must pass the approval process before they can be used. To further simplify the calibration process, we developed a third method that relies solely on material and devices already approved for use in the OR.

We use the same principle that the ultrasound probe must be calibrated w.r.t. a fixed point in space in relation to the ultrasound plane. We then create a model of the ultrasound probe by imaging the probe in a CT scanner and extracting a surface model by segmenting the image. Fig. D.2.3 shows a laparoscopic probe and the surface model of its tip derived from a CT scan of the probe.. Three easy-to-identify landmarks (e.g. for laparoscopic probe: tip of the probe, entrance of the biopsy channel and lower exit of biopsy channel) are approximately identified in the model. The user is later instructed to touch these landmarks to initialize the registration process. We inserted a needle in the biopsy channel to give the model more spatial extent. We segmented the needle and the probe separately. The model with the needle inserted is used to guide the registration algorithm in the initial phase of the calibration process.

Preoperative Calibration:

The preoperative calibration involves a calibration phantom, two EM sensors, and the probe to be calibrated. One of the sensors is mounted on the probe and the second one is used to scan the surface of the probe. In addition to the coordinate systems defined at the beginning of Section D.2 we also have the coordinate system of the scanning sensor which we denote by \mathbf{C}_S . The fixed coordinate system \mathbf{C}_F ,

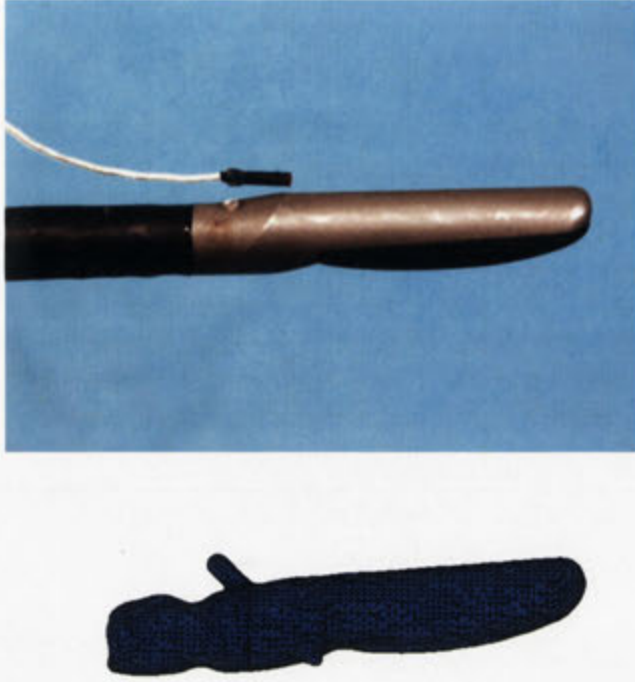


Figure D.2: A laparoscopic probe and its 3D mesh model (needle partially visible in the biopsy channel).

in this setup is defined at an arbitrary (but fixed) position in the segmented CT volume.

We first determine the calibration matrix between the probe sensor and the ultrasound plane ${}^S\mathbf{T}_U$ using the single-wall phantom. We then scan the surface of the probe by moving the second sensor slowly against the surface of the probe. The scanning sensor is attached to a stylus for easy handling.

The precise location of the coordinate system attached to the scanning sensor is not known. The offset between the location of the coordinate system and the tip of the sensor can be described by a translation. The position of the tip of the sensor \mathbf{x}_t based on the sensor measurements $\mathbf{x}_{s'}$ is given by

$$\mathbf{x}_t = \mathbf{x}_{s'} + [v_x \ v_y \ v_z]^T, \quad (\text{D.3})$$

where the unknown translation vector $\mathbf{v} = [v_x \ v_y \ v_z]^T$ is computed as part of our calibration/registration process.

The set of points measured on the surface of the probe $\{\mathbf{x}_t\}$ are related to corresponding surface points in the model $\{\mathbf{x}_m\}$ by a rigid transformation. The

iterative closest point (ICP) algorithm can be used to register the two point sets as long as \mathbf{v} is given. The adaptation of ICP to solve for \mathbf{v} in addition to the rigid registration parameters is not trivial. One could install the sensor on a stylus and calibrate the stylus to retrieve \mathbf{v} . We did not find this option convenient nor particularly helpful for overall accuracy. The error is further compounded by the stylus calibration. Alternatively, we solve for \mathbf{v} and the rigid registration together using a 9-parameter local optimization algorithm with a closest point cost function (a suitable optimization algorithm such as Powell, Simplex, or Gradient Descent variants can be used). The outcome of the closest point optimization is the transformation matrix from the tracker coordinates to the model (fixed) coordinates ${}^F\mathbf{T}_T$. We also measure the position of the sensor attached to the probe and can now compute ${}^F\mathbf{T}_U$ using (D.1). Note that the probe must remain stationary during the scanning process for this method to work. However, it is more convenient to have the flexibility to move the probe to gain access to the surface. The ability to move the probe has the added advantage that one does not have to worry about securing the probe in position and errors due to pressure against the tip of the probe which may cause small movements. To this end, we record the position of the scanning sensor and the probe sensor simultaneously, and compute the position of the surface points in the coordinate system of the probe sensor:

$$\mathbf{x}_s = {}^T\mathbf{T}_S^{-1} \mathbf{x}_t \quad (\text{D.4})$$

Using $\{\mathbf{x}_s\}$ instead of $\{\mathbf{x}_t\}$ means that the outcome of the registration process is ${}^F\mathbf{T}_S$ and (D.1) can be simplified to

$${}^F\mathbf{T}_U = {}^F\mathbf{T}_S {}^S\mathbf{T}_U. \quad (\text{D.5})$$

Intraoperative Calibration:

The intraoperative calibration involves two EM sensors and the probe to be calibrated. The calibration process is similar to the preoperative calibration except that no phantom-based calibration is involved. The surface of the probe is scanned using a sensor and the resulting object measurements are registered against the model to determine ${}^F\mathbf{T}_S$, as before. Since ${}^F\mathbf{T}_U$ is known, the calibration matrix is computed using:

$${}^S\mathbf{T}_U = {}^F\mathbf{T}_S^{-1} {}^F\mathbf{T}_U. \quad (\text{D.6})$$

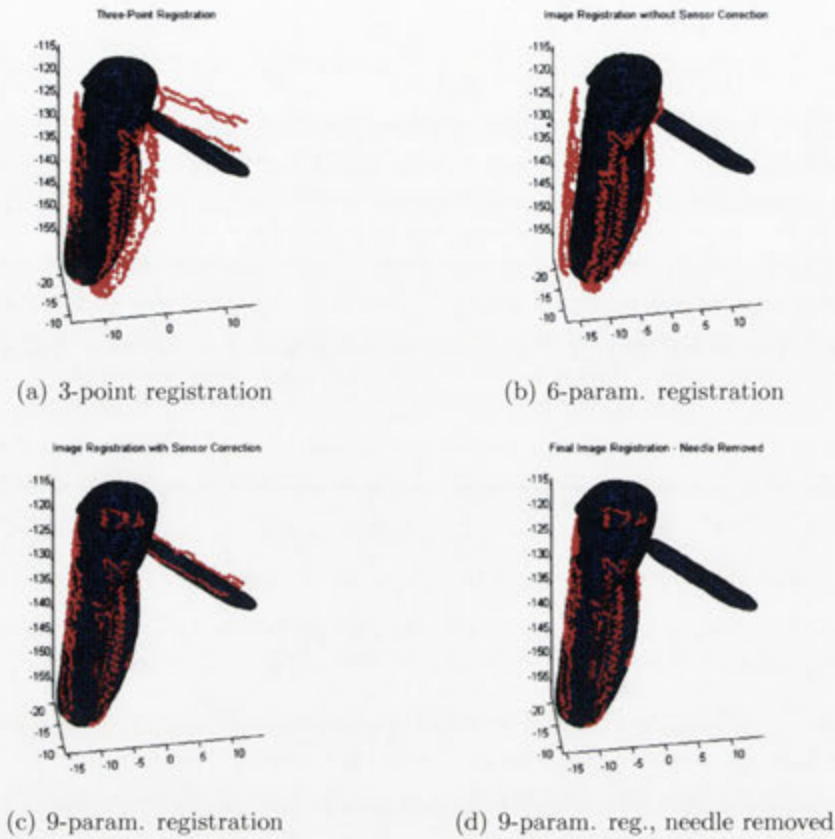


Figure D.3: Incremental improvement in the alignment of a scanned probe (red cloud) with the model (blue): (a) the registration is initialized with a 3-point rigid alignment first, (b) a 6-parameter registration is unable to register the point cloud to the surface of the model due to the distance of the origin of the sensor's coordinate system from its tip, (c) a 9-parameter optimization algorithm retrieves rigid registration parameters together with the sensor's calibration, (d) starting from the results of the previous run, a second 9-parameter optimization is performed with the needle points removed for improved alignment.

Object to Model Registration:

To ensure convergence, a three-stage registration is performed for object to model registration. Each stage is designed to improve the registration accuracy and is initialized from the solution returned by the previous stage.

1. **3-point registration:** the user is requested to identify 3 previously selected landmarks in a pre-defined order by touching the corresponding points on the object by the sensor. An approximate rigid transformation from the object to the model is computed and used for initializing the registration optimization algorithm.
2. **9-parameter registration - initial:** a 9-parameter registration consisting

of rigid object-to-model alignment parameters (6 parameters) and position of the sensor's tip in the coordinate system of the sensor (3 parameters) is performed. The results are used to initialize the next registration stage.

3. **9-parameter registration - final:** data points that belong to the needle are removed from both the object samples and the model and a constrained 9-parameter optimization is performed to determine the registration parameters more accurately.

Fig. D.3 illustrates incremental improvement in registration of a point of clouds measured from the surface of a laparoscopic probe (shown in red) to a model of the probe (shown in blue). Fig. D.3(b) shows the result of a 6-parameter registration which does not include the calibration parameters of the sensor. This is shown for comparison with 9-parameter registrations only and is not part of our registration algorithm.

D.3 Results

An LUS probe was calibrated preoperatively using the single-wall phantom. The probe was then registered to its 3D segmented model to compute the calibration matrix w.r.t. a fixed point in the model. For the intraoperative calibration we used a different sensor which was placed at a different location on the surface of the probe. The calibration was determined by the closet point calibration algorithm. Table D.1 shows air calibration precision for 6 experiments. The first three rows show the mean registration error for each registration step. The registration error is reduced by each step. To validate the air calibration method, the intraoperative sensor was also independently calibrated using the single-wall phantom so that the air calibration results can be compared with the single-wall phantom. Single-wall calibration was also performed several times. The results are summarized in Table D.2. The precision of the calibration methods was computed for a point in the center of the ultrasound image. This makes sense as the operators tend to keep objects of interests in the center of the field of view. We also report the preoperative calibration precision with the single-wall phantom for completeness.

D.4 Discussion

This study demonstrates a fast, easy-to-use method for instrument calibration suitable for intraoperative use. It uses equipment and material already available in the

Table D.1: Registration error and the calibration precision for a number of experiments given in mm

Experiment	1	2	3	4	5	6
Mean registration error (step 1)	2.14	1.60	1.97	3.20	1.88	2.81
Mean registration error (step 2)	0.53	0.48	0.59	0.60	0.81	0.64
Mean registration error (step 3)	0.50	0.48	0.59	0.54	0.73	0.63
Air calibration error	2.07	2.33	0.37	2.18	2.27	2.20

Table D.2: Precision of the single-wall phantom and the closest point calibration methods in mm

Method	Mean	Std. Dev.	Min	Max
Wall-phantom calib., pre-op. sensor	1.26	0.44	0.79	1.85
Wall-phantom calib., intra-op sensor	1.17	0.40	0.66	1.71
Air calib., intra-op sensor	1.90	0.76	0.37	2.33

OR. Our experiments were not directed toward reconstruction of 3D freehand ultrasound volumes but with approximate alignment of the B-mode ultrasound with a preoperative CT. The reformatted CT were shown side-by-side with the ultrasound stream to provide anatomical context for interpreting the ultrasound and improving the navigation of laparoscopic and endoscopic ultrasound [123]. For this application, the highest calibration accuracy was not the primary concern and we limited ourselves to qualitative assessment of the resulting calibration. It will be interesting to investigate the limits of the proposed methods for freehand ultrasound and to provide accuracy results in addition to precision in the future work. Single-wall phantom preoperative calibration is easy to perform but not the most accurate method. We expect the overall precision and accuracy to improve with a better preoperative calibration method.

Bibliography

- [1] J. West, J. M. Fitzpatrick, M. Y. Wang, B. M. Dawant, C. R. Maurer, Jr., R. M. Kessler, R. J. Maciunas, C. Barillot, D. Lemoine, A. Collignon, F. Maes, P. Suetens, D. Vandermeulen, P. A. van den Elsen, S. Napel, T. S. Sumanaweera, B. Harkness, P. F. Hemler, D. L. G. Hill, D. J. Hawkes, C. Studholme, J. B. A. Maintz, M. A. Viergever, G. Malandain, X. Pennec, M. E. Noz, G. Q. Maguire, Jr., M. Pollack, C. A. Pelizzari, R. A. Robb, D. Hanson, and R. P. Woods, "Comparison and evaluation of retrospective intermodality brain image registration techniques," *Journal of Computer Assisted Tomography*, vol. 21, no. 4, pp. 554–566, 1997.
- [2] L. G. Brown, "A survey of image registration techniques," *ACM Computing Surveys*, vol. 24, no. 4, pp. 325–376, Dec. 1992.
- [3] J. B. A. Maintz and M. A. Viergever, "A survey of medical image registration," *Med. Image Anal.*, vol. 2, no. 1, pp. 1–36, 1998.
- [4] J. P. W. Pluim, J. B. A. Maintz, and M. A. Viergever, "Mutual-information-based registration of medical images: A survey," *IEEE Trans. on Med. Imaging*, vol. 22, no. 8, pp. 986–1004, Aug. 2003.
- [5] J. Modersitzki, *Mumerical Methods for Image Registration*. New York: Oxford University Press, 2004.
- [6] R. Shams, P. Sadeghi, R. A. Kennedy, and R. I. Hartley, "A survey of medical image registration on multicore and the GPU," *IEEE Signal Processing Mag.* (to appear), Mar. 2010.
- [7] D. Rueckert, L. I. Sonoda, C. Hayes, D. L. G. Hill, M. O. Leach, and D. J. Hawkes, "Nonrigid registration using free-form deformations: Application to breast MR images," *IEEE Trans. on Med. Imaging*, vol. 18, no. 8, pp. 712–721, Aug. 1999.

- [8] C. R. Meyer, J. L. Boes, B. Kim, P. H. Bland, K. R. Zasadny, P. V. Kison, K. Koral, K. A. Frey, and R. L. Wahl, "Demonstration of accuracy and clinical versatility of mutual information for automatic multimodality image fusion using affine and thin-plate spline warped geometric deformations." *Med. Image Anal.*, vol. 1, no. 3, pp. 195–206, 1997.
- [9] G. Soza, M. Bauer, P. Hastreiter, C. Nimsy, and G. Greiner, "Non-rigid registration with use of hardware-based 3D Bézier functions," in *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2002, pp. 549–556.
- [10] T. M. Lehmann, C. Gönnér, and K. Spitzer, "Survey: Interpolation methods in medical image processing," *IEEE Trans. on Med. Imaging*, vol. 18, no. 11, pp. 1049–1075, Nov. 1999.
- [11] C. R. Castro-Pareja, J. M. Jagadeesh, and R. Shekhar, "FAIR: a hardware architecture for real-time 3-D image registration," *IEEE Trans. on Info. Technology in Biomedicine*, vol. 7, no. 4, pp. 426–434, Dec. 2003.
- [12] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge, UK: Cambridge University Press, 2003.
- [13] W. Wein, "Multimodal integration of medical ultrasound for treatment planning and interventions," Ph.D. dissertation, Technical University of Munich, 2007.
- [14] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423/623–656, 1948.
- [15] A. Collignon, F. Maes, D. Delaere, D. Vandermeulen, P. Suetens, and G. Marchal, "Automated multimodality medical image registration using information theory," in *Proc. Int. Conf. Information Processing in Med. Imaging: Computational Imaging and Vision 3*, Apr. 1995, pp. 263–274.
- [16] P. Viola and W. M. Wells III, "Alignment by maximization of mutual information," in *Proc. Int. Conf. Computer Vision (ICCV)*, Jun. 1995, pp. 16–23.
- [17] C. Studholme, D. L. G. Hill, and D. J. Hawkes, "An overlap invariant entropy measure of 3D medical image alignment," in *Pattern Recognit.*, vol. 32, 1999, pp. 71–86.

- [18] F. Maes, A. Collignon, D. Vandermeulen, G. Marchal, and P. Suetens, "Multimodality image registration by maximization of mutual information," *IEEE Trans. Med. Imaging*, vol. 16, no. 2, pp. 187–198, Apr. 1997.
- [19] A. Roche, G. Malandain, X. Pennec, and N. Ayache, "The correlation ratio as a new similarity measure for multimodal image registration," in *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, Oct. 1998, pp. 1115–1124.
- [20] —, "Multimodal image registration by maximization of the correlation ratio," INRIA, Tech. Rep. 3378, Mar. 1998.
- [21] R. Shams, R. A. Kennedy, P. Sadeghi, and R. Hartley, "Gradient intensity-based registration of multi-modal images of the brain," in *Proc. IEEE Int. Conf. on Computer Vision (ICCV)*, Rio de Janeiro, Brazil, Oct. 2007.
- [22] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. New York, NY, USA: Cambridge University Press, 2007.
- [23] J. A. Nedler and R. Mead, "A simplex method for function minimization," *Computer Journal*, vol. 7, pp. 308–331, 1965.
- [24] J. E. Dennis Jr and V. Torczon, "Direct search methods on parallel machines," *SIAM Journal on Optimization*, vol. 1, pp. 448–474, 1991.
- [25] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, "Lipschitzian optimization without the Lipschitz constant," *J. Optim. Theory Appl.*, vol. 79, no. 1, pp. 157–181, 1993.
- [26] E. Cantú-Paz, "A survey of parallel genetic algorithms," *CALCULATEURS PARALLELES*, vol. 10, 1998.
- [27] T. Butz and J.-P. Thiran, "Affine registration with feature space mutual information," in *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2001, pp. 549–556.
- [28] M. P. Wachowiak and T. M. Peters, "High-performance medical image registration using new optimization techniques," *IEEE Transactions on Information Technology in Biomedicine*, vol. 10, no. 2, pp. 344–353, Apr. 2006.

- [29] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. New York, NY, USA: Cambridge University Press, 2007, ch. 10.
- [30] R. P. Brent, *Algorithms for Minimization without Derivatives*, 2nd ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 1973.
- [31] R. Shams, P. Sadeghi, and R. A. Kennedy, "Gradient intensity: A new mutual information based registration method," in *Proc. IEEE Computer Vision and Pattern Recognition (CVPR) Workshop on Image Registration and Fusion*, Minneapolis, MN, Jun. 2007.
- [32] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. New York, NY, USA: Cambridge University Press, 2007, ch. 7.
- [33] J. P. Keener, *Principles of Applied Mathematics: Transformation and Approximation*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1988.
- [34] J. P. W. Pluim, J. B. A. Maintz, and M. A. Viergever, "Image registration by maximization of combined mutual information and gradient information," *IEEE Trans. on Med. Imaging*, vol. 19, no. 8, pp. 809–814, Aug. 2000.
- [35] J. Liu, J. Tian, and Y. Dai, "Multi-modal medical image registration based on adaptive combination of intensity and gradient field mutual information," in *IEEE Int. Conf. of Engineering in Medicine and Biology Society (EMBS)*, Aug. 2006, pp. 1429–1432.
- [36] I. Sobel, "Neighbourhood coding of binary images fast contour following and general array binary processing," *Computer Graphics and Image Processing*, vol. 8, pp. 127–135, 1978.
- [37] B. K. P. Horn, *Robot Vision*. The Massachusetts Institute of Technology: MIT Press, 1986.
- [38] P. V. C. Hough, "A method and means for recognizing complex patterns," in *U.S. Patent No. 3,069,654*, 1962.
- [39] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Comm. ACM*, vol. 15, pp. 11–15, Jan. 1972.

- [40] A. Rajwade, A. Banerjee, and A. Rangarajan, "A new method for probability density estimation with application to mutual information based image registration," in *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)*, Jun. 2006.
- [41] *Brain Web*. <http://www.bic.mni.mcgill.ca/brainweb/>: Montreal Neurological Institute, McGill University, 2006.
- [42] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C*, 2nd ed. Cambridge: Cambridge University Press, 1992.
- [43] A. Cole-Rhodes, K. Johnson, J. LeMoigne, and I. Zavorin, "Multiresolution registration of remote sensing imagery by optimization of mutual information using a stochastic gradient," *IEEE Trans. on Image Processing*, vol. 12, no. 12, pp. 1495–1511, Dec. 2003.
- [44] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004.
- [45] R. Shams, R. A. Kennedy, and P. Sadeghi, "Efficient image registration by decoupled parameter estimation using gradient-based techniques and mutual information," in *Proc. IEEE Region 10 Conf. (TENCON)*, Taipei, Taiwan, Oct. 2007.
- [46] *Retrospective Image Registration Evaluation Project*, <http://insight-journal.org/rire/>, 2007.
- [47] *OpenMP Application Programming Interface, version 3.0*. <http://openmp.org/wp/openmp-specifications/>: OpenMP, 2009.
- [48] B. Chapman, G. Jost, and R. van der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming*. Cambridge, Massachusetts, USA: The MIT Press, 2008.
- [49] T. Rohlfing and C. R. Maurer, Jr., "Nonrigid image registration in shared-memory multiprocessor environments with application to brains, breasts, and bees," *IEEE Trans. on Info. Technology in Biomedicine*, vol. 7, no. 1, pp. 16–25, Mar. 2003.
- [50] E. L. W. Gropp and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message Passing Interface*, 2nd ed. Cambridge, MA, USA: MIT Press, 1999.

- [51] F. Ino, K. Ooyama, and K. Hagihara, "A data distributed parallel algorithm for nonrigid image registration," *Parallel Computing*, vol. 31, no. 1, pp. 19–43, Jan. 2005.
- [52] S. Ourselin, R. Stefanescu, and X. Pennec, "Robust registration of multimodal images: Towards real-time clinical applications," in *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2002, pp. 140–147.
- [53] M. Ohara, H. Yeo, F. Savino, G. Iyengar, L. Gong, H. Inoue, H. Komatsu, V. Sheinin, and S. Daijavad, "Accelerating mutual-information-based linear registration on the Cell broadband engine processor," in *IEEE Int. Conf. on Multimedia and Expo*, 2007, pp. 272–275.
- [54] M. Ohara, H. Yeo, F. Savino, G. Iyengar, L. Gong, H. Inoue, H. Komatsu, V. Sheinin, S. Daijavad, and B. Erickson, "Real-time mutual-information-based linear registration on the Cell broadband engine processor," in *IEEE Int. Symp. on Biomedical Imaging (ISBI)*, 2007, pp. 33–36.
- [55] J. Rohrer and L. Gong, "Accelerating mutual information based 3D non-rigid registration using the Cell/B.E. processor," in *Proc. Workshop on Cell Systems and Applications (WCSA)*, 2008, pp. 32–40.
- [56] C. R. Castro-Pareja, J. M. Jagadeesh, and R. Shekhar, "FPGA-based acceleration of mutual information calculation for real-time 3D image registration," in *Proc. SPIE Medical Imaging: Image Processing*, 2008.
- [57] O. Dandekar and R. Shekhar, "FPGA-accelerated deformable image registration for improved target-delineation during CT-guided interventions," *IEEE Trans. on Biomedical circuits and systems*, vol. 1, no. 2, pp. 116–127, Jun. 2007.
- [58] G. C. Sharp, N. Kandasamy, H. Singh, and M. Folkert, "GPU-based streaming architectures for fast cone-beam CT image reconstruction and demons deformable registration," *Phys. Med. Biol.*, vol. 52, no. 19, pp. 5771–5783, 2007.
- [59] R. Shams and N. Barnes, "Speeding up mutual information computation using NVIDIA CUDA hardware," in *Proc. Digital Image Computing: Techniques and Applications (DICTA)*, Adelaide, Australia, Dec. 2007, pp. 555–560.

- [60] Y. Lin and G. Medioni, "Mutual information computation and maximization using GPU," in *Proc. IEEE Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2008, pp. 1–6.
- [61] W. Plishker, O. Dandekar, S. S. Bhattacharyya, and R. Shekhar, "Towards systematic exploration of tradeoffs for medical image registration on heterogeneous platforms," in *IEEE Biomedical Circuits and Systems Conference*, Nov. 2008, pp. 53–56.
- [62] P. Muyan-Özçelik, J. D. Owens, J. Xia, and S. S. Samant, "Fast deformable registration on the GPU: A CUDA implementation of demons," in *Proc. Int. Conf. on Computational Science and Its Applications (ICCSA)*, 2008, pp. 5–8.
- [63] R. Shams, P. Sadeghi, R. A. Kennedy, and R. Hartley, "Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images," *Computer Methods and Programs in Biomedicine (accepted)*, Nov. 2009.
- [64] A. Ruiz, M. Ujaldon, L. Cooper, and K. Huang, "Non-rigid registration for large sets of microscopic images on graphics processors," *Journal of Signal Processing Systems*, vol. 55, no. 1-3, pp. 229–250, Apr. 2009.
- [65] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, p. 114117, Apr. 1965.
- [66] *Compute Unified Device Architecture (CUDA) Programming Guide, version 2.2*. <http://developer.nvidia.com/object/cuda.html>: NVIDIA, 2009.
- [67] R. Strzodka, M. Droske, and M. Rumpf, "Image registration by a regularized gradient flow. a streaming implementation in DX9 graphics hardware," *Computing*, vol. 73, no. 4, pp. 373–389, Nov. 2004.
- [68] A. Khamene, R. Chisu, W. Wein, N. Navab, and F. Sauer, "A novel projection based approach for medical image registration," in *Third International Workshop on Biomedical Image Registration (WBIR)*, Utrecht, The Netherlands, Jun. 2006, pp. 247–256.
- [69] F. Ino, J. Gomita, Y. Kawasaki, and K. Hagihara, "A GPGPU approach for accelerating 2-D/3-D rigid registration of medical images," in *Parallel and Distributed Processing and Applications*, Feb. 2006.

- [70] C. Vetter, C. Guetter, C. Xu, and R. Westermann, "Non-rigid multi-modal registration on the GPU," in *Proc. SPIE Medical Imaging: Image Processing*, Feb. 2007.
- [71] Z. Fan, C. Vetter, C. Guetter, D. Yu, R. Westermann, A. Kaufman, and C. Xu, "Optimized GPU implementation of learning-based non-rigid multi-modal registration," in *Proc. SPIE Medical Imaging: Image Processing*, 2008.
- [72] N. Courty and P. Hellier, "Accelerating 3D non-rigid registration using graphics hardware," *International Journal of Image and Graphics*, vol. 8, no. 1, pp. 1–18, Jan. 2008.
- [73] A. Kubias, F. Deinzer, T. Feldmann, S. Paulus, D. Paulus, B. Schreiber, and T. Brunner, "2D/3D image registration on the GPU," *Pattern Recognition and Image Analysis*, vol. 18, no. 3, pp. 381–389, Sep. 2008.
- [74] *ATI stream computing user guide, version 1.4.0.a*. <http://developer.amd.com/>: ATI, 2009.
- [75] R. Shams and R. A. Kennedy, "Efficient histogram algorithms for NVIDIA CUDA compatible devices," in *Proc. Int. Conf. on Signal Processing and Communications Systems (ICSPCS)*, Gold Coast, Australia, Dec. 2007, pp. 418–422.
- [76] *NVIDIA CUDA C Programming Best Practices Guide*. <http://developer.nvidia.com/object/cuda.html>: NVIDIA, 2009.
- [77] A. El Zein, E. McCreath, A. Rendell, and A. Smola, "Performance evaluation of the NVIDIA GeForce 8800 GTX GPU for machine learning," in *Proc. Int. Conf. on Computational Science (ICCS)*, 2008, pp. 466–475.
- [78] J. W. Sheaffer, D. P. Luebke, and K. Skadron, "A hardware redundancy and recovery mechanism for reliable scientific computation on graphics processors," in *Graphics Hardware*, T. Aila and M. Segal, Eds., Aug. 2007, pp. 55–64.
- [79] S. Warfield, F. Jolesz, and R. Kikinis, "A high performance computing approach to the registration of medical imaging data," *Parallel Computing*, vol. 24, no. 9–10, pp. 1345–1368, Sep. 1998.
- [80] A. Köhn, J. Drexler, F. Ritter, M. König, and H. O. Peitgen, "GPU accelerated image registration in two and three dimensions," in *Bildverarbeitung für die Medizin*, 2006, pp. 261–265.

- [81] D. W. Scott, "On optimal and data-based histograms," *Biometrika*, vol. 66, no. 3, pp. 605–610, Dec. 1979.
- [82] K. H. Knuth, "Optimal data-based binning for histograms," *ArXiv Physics e-prints*, May 2006.
- [83] V. Podlozhnyuk, "64-bin histogram," NVIDIA, Tech. Rep., 2007.
- [84] O. Fluck, S. Aharon, D. Cremers, and M. Rousson, "Gpu histogram computation," in *SIGGRAPH Research posters*, Jul. 2006.
- [85] T. Scheuermann and J. Hensley, "Efficient histogram generation using scattering on GPUs," in *Proce. Symposium on Interactive 3D Graphics and Games*, Apr. 2007.
- [86] D. E. Knuth, *The Art of Computer Programming*. Addison-Wesley, 1973, vol. 3 - Sorting and Searching.
- [87] K. E. Batcher, "Sorting networks and their applications," in *Proc. AFIPS Spring Joint Comput. Conf.*, vol. 32, 1968, pp. 307–314.
- [88] F. Maes, D. Vandermeulen, and P. Suetens, "Comparative evaluation of multiresolution optimization strategies for multimodality image registration by maximization of mutual information," *Med. Image Anal.*, vol. 3, no. 4, pp. 373–386, 1999.
- [89] P. Thevenaz and M. Unser, "Optimization of mutual information for multi-resolution image registration," *IEEE Trans. Image Processing*, vol. 9, no. 12, pp. 2083–2099, Dec. 2000.
- [90] D. Mattes, D. R. Haynor, H. Vesselle, T. K. Lewellen, and W. Eubank, "PET-CT image registration in the chest using free-form deformations," *IEEE Trans. on Medical Imaging*, vol. 22, no. 1, pp. 120–128, Jan. 2003.
- [91] H. Maul, A. Scharf, P. Baier, M. Wüstemann, H. H. Günter, G. Gebauer, and C. Sohn, "Ultrasound simulators: experience with the SonoTrainer and comparative review of other training systems," *Ultrasound Obstet. Gynecol.*, vol. 24, no. 5, pp. 581–585, Oct. 2004.
- [92] D. Aiger and D. Cohen-Or, "Real-time ultrasound imaging simulation," *Real-Time Imag.*, vol. 4, no. 4, pp. 263–274, 1998.

- [93] H.-H. Ehrlicke, "SONOSim3D: a multimedia system for sonography simulation and education with an extensible case database," *European J. of Ultrasound*, vol. 7, pp. 225–300, 1998.
- [94] D. Henry, J. Troccaz, J. L. Bosson, and O. Pichot, "Ultrasound imaging simulation: Application to the diagnosis of deep venous thromboses of lower limbs," in *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, Oct. 1998, pp. 1032–1040.
- [95] J. Stallkamp and M. Wapler, "UltraTrainer: a training system for medical ultrasound examination," in *Medicine Meets Virtual Reality (MMVR)*, Jan. 1998.
- [96] M. Weidenbach, C. Wick, S. Pieper, K. J. Quast, T. Fox, G. Grunst, and D. A. Redel, "Augmented reality simulator for training in two-dimensional echocardiography," in *Computers and Biomedical Research*, vol. 33, 2000, pp. 11–22.
- [97] C. Terkamp, G. Kirchner, J. Wedemeyer, A. Dettmer, J. Kielstein, H. Rein-dell, J. Bleck, M. Manns, and M. Gebel, "Simulation of abdomen sonography. evaluation of a new ultrasound simulator," *Ultraschall in Med*, vol. 24, pp. 239–244, 2003.
- [98] I. M. Heer, K. Middendorf, S. Müller-Egloff, M. Dugas, and A. Strauss, "Ultrasound training: the virtual patient," *Ultrasound Obstet. Gynecol.*, vol. 24, pp. 440–444, 2004.
- [99] A. M. Tahmasebi, K. Hashtrudi-Zaad, D. Thompson, and P. Abolmaesumi, "A framework for the design of a novel haptic-based medical training simulator," *IEEE Transactions on Information Technology in Biomedicine*, 2008.
- [100] *UltraSim: Ultrasound training simulator*. <http://www.medsim.com/>: Med-Sim Advanced Medical Simulations, Ltd., 2008.
- [101] A. Hostettler, C. Forest, A. Forgione, L. Soler, and J. Marescaux, "Real-time ultrasonography simulator based on 3D CT-scan images," in *Medicine Meets Virtual Reality (MMVR)*, vol. 111, Feb. 2005, pp. 191–193.
- [102] F. P. Vidal, N. W. John, A. E. Healey, and D. A. Gould, "Simulation of ultrasound guided needle puncture using patient specific data with 3D textures and volume haptics," *Comp. Anim. Virtual Worlds*, 2007.

- [103] W. Wein, A. Khamene, D. Clevert, O. Kutter, and N. Navab, "Simulation and fully automatic multimodal registration of medical ultrasound," in *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, Oct. 2007, pp. 136–143.
- [104] R. Shams, R. Hartley, and N. Navab, "Real-time simulation of medical ultrasound from CT images," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, New York, USA, Sep. 2008, pp. 734–741.
- [105] J. A. Jensen, "Field: A program for simulating ultrasound systems," in *10th Nordic-Baltic Conference on Biomedical Imaging Published in Medical & Biological Engineering & Computing*, 1996, pp. 351–353.
- [106] J. A. Jensen and N. B. Svendsen, "Calculation of pressure fields from arbitrarily shaped, apodized, and excited ultrasound transducers," *IEEE Trans. Ultrason.*, vol. 39, pp. 262–267, 1992.
- [107] J. A. Jensen and S. I. Nikolov, "Fast simulation of ultrasound images," in *IEEE Ultrasonics Symposium*, 2000, pp. 1721–1724.
- [108] G. E. Tupholme, "Generation of acoustic pulses by baffled plane pistons," *Mathematika*, vol. 16, pp. 209–224, 1969.
- [109] P. R. Stepanishen, "Transient radiation from pistons in an infinite planar baffle," *J. Acoust. Soc. Am.*, vol. 49, pp. 1629–1638, 1971.
- [110] J. A. Jensen, "Simulation of advanced ultrasound systems using Field II," in *IEEE Int. Symp. on Biomedical Imaging (ISBI)*, Apr. 2004, pp. 636–639.
- [111] W. Hedrick, D. Hykes, and D. Starchman, *Ultrasound Physics and Instrumentation*, 3rd ed. Mosby - Year Book, Inc., 1995.
- [112] O. Monga, R. Deriche, G. Malandain, and J. P. Cocquerez, "Recursive filtering and edge tracking: two primary tools for 3D edge detection," *Image and Vision Computing*, vol. 9, no. 4, pp. 203–214, Aug. 1991.
- [113] J. A. Jensen, "Linear description of ultrasound imaging systems," Technical Univ. of Denmark, Ørsted, Denmark, Jul. 1999.
- [114] K. Engel, M. Hadwiger, J. Kniss, and C. Rezk-Salama, *Real-Time Volume Graphics*. AK Peters, Ltd., 2006.

- [115] P. Sabella, "A rendering algorithm for visualizing 3D scalar fields," *Computer Graphics*, vol. 22, no. 4, pp. 51–58, 1988.
- [116] J. Kruger and R. Westermann, "Acceleration Techniques for GPU-based Volume Rendering," *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, 2003.
- [117] S. Stegmaier, M. Strengert, T. Klein, and T. Ertl, "A Simple and Flexible Volume Rendering Framework for Graphics-Hardware-based Raycasting," *Proceedings of the International Workshop on Volume Graphics*, vol. 5, pp. 187–195, 2005.
- [118] H. Scharsach, M. Hadwiger, A. Neubauer, S. Wolfsberger, and K. Buhler, "Perspective Isosurface and Direct Volume Rendering for Virtual Endoscopy Applications," *Proceedings of Eurovis/IEEE-VGTC Symposium on Visualization*, pp. 315–322, 2006.
- [119] K. Engel, M. Kraus, and T. Ertl, "High-quality pre-integrated volume rendering using hardware-accelerated pixel shading," *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pp. 9–16, 2001.
- [120] Q. Zhang, R. Eagleson, and T. Peters, "Rapid Voxel Classification Methodology for Interactive 3D Medical Image Visualization," in *MICCAI 2007 Proceedings*, ser. Lecture Notes in Computer Science. Springer, Oct. 2007.
- [121] J. Ellsmere, J. Stoll, D. W. Rattner, D. Brooks, R. Kane, W. M. Wells III, R. Kikinis, and K. G. Vosburgh, "A navigation system for augmenting laparoscopic ultrasound," in *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, Nov. 2003, pp. 184–191.
- [122] J. Ellsmere, J. Stoll, W. M. Wells III, R. Kikinis, K. G. Vosburgh, R. Kane, D. Brooks, and D. Rattner, "A new visualization technique for laparoscopic ultrasound," *Surgery*, vol. 136, no. 1, pp. 84–92, Jul. 2004.
- [123] K. G. Vosburgh, N. Stylopoulos, R. San José Estépar, R. E. Ellis, E. Samset, and C. C. Thompson, "EUS with CT improves efficiency and structure identification over conventional EUS," *Gastrointestinal Endoscopy*, vol. 65, no. 6, pp. 866–870, 2007.

- [124] R. San José Estépar, N. Stylopoulos, R. E. Ellis, E. Samset, C.-F. Westin, C. C. Thompson, and K. G. Vosburgh, "Towards scarless surgery: An endoscopic ultrasound navigation system for transgastric access procedures," *Computer Aided Surgery*, vol. 12, no. 6, pp. 311–324, 2007.
- [125] R. Prager, R. Rohling, A. Gee, and L. Berman, "Rapid calibration for 3-D free-hand ultrasound," *Ultrasound in Med. & Biol.*, vol. 24, no. 6, pp. 855–869, Jul. 1998.
- [126] T. K. Chen, A. D. Thurston, R. E. Ellis, and P. Abolmaesumi, "A real-time freehand ultrasound calibration system with automatic accuracy feedback and control," *Ultrasound in Med. & Biol.*, vol. 35, no. 1, pp. 79–93, Jan. 2009.
- [127] L. Mercier, T., Langø, F. Lindseth, and L. D. Collins, "A review of calibration techniques for freehand 3D ultrasound systems," *Ultrasound in Med. & Biol.*, vol. 31, no. 4, pp. 449–471, Apr. 2005.
- [128] E. M. Boctor, I. Iordachita, G. Fichtinger, and G. D. Hager, "Ultrasound self-calibration," in *Proc. SPIE Medical Imaging: Visualization, Image-Guided Procedures, and Display*, Feb. 2006.
- [129] W. Wein and A. Khamene, "Image-based method for in-vivo freehand ultrasound calibration," in *Proc. SPIE Medical Imaging: Image Processing*, Feb. 2008.
- [130] E. M. Boctor, A. Jain, M. A. Choti, R. H. Taylor, and G. Fichtinger, "Rapid calibration method for registration and 3D tracking of ultrasound images using spatial localizer," in *Proc. SPIE Medical Imaging: Visualization, Image-Guided Procedures, and Display*, Feb. 2003, pp. 521–532.